

Module graphique Tkinter pour Python

LE module graphique de référence de **python** est Tkinter. Il en existe d'autres: [wxPython](#), [PyQt](#) ou [PyGTK](#). Cette page comporte des répétitions à l'attention des programmeurs pressés qui brûlent les étapes. Les exemples, testés sur GNU/Linux Debian, python 2.7/3.2 et Tkinter 8.5, sont tous concrets et fonctionnels.

Cette page a été commencée en début 2007 pour me servir de bloc-note personnel, elle évolue donc surtout en fonction de mes nécessités. Elle est néanmoins écrite dans un souci de servir à d'autres: pour plus de clarté, les variables sont écrites en minuscules et terminées par un chiffre ; les explicitations des paramètres sont regroupées dans une liste à puces qui suit chaque exemple.

Spinbox, **Scale** et **OptionMenu** ont été ajoutés cet automne 2014 (mais pas encore **Menubutton**) ; le puissant **Text**, qui mériterait un chapitre à lui tout seul, commence à être plus explicite.

1. Utilisation de Tkinter	3. L'espace Canvas	5. Modules	7. Modules associés
1.1 Charger le module	3.1 Texte graphique	5.1 <code>mainloop</code> / <code>quit</code>	7.1 Boîtes à message
1.2 Ouvrir une fenêtre	3.2 Ligne et point	5.2 <code>pack</code> / <code>grid</code> / <code>place</code>	7.2 Boîtes de saisie
1.3 Lancer le script	3.3 Surfaces	5.3 <code>config</code>	7.3 Sélecteurs de fichier
2. Widgets simples	3.4 Afficher un fichier image	5.4 <code>destroy</code>	7.4 Sélecteurs de couleur
2.1 Frame et LabelFrame	3.5 Afficher une image bitmap	5.5 <code>bind</code> (événements)	A. Documentation
2.2 Label	3.6 Reconfigurer ou supprimer	5.6 <code>after</code> (temps)	A.1 dans le logiciel
2.3 Texte éditable	4. Widgets complexes	6. Styles	A.2 sur GNU/Linux
2.4 Entry	4.1 OptionMenu ^{Nv}	6.1 Alignements	A.3 sur Internet
2.5 Button	4.2 PanedWindow	6.2 Reliefs	A.4 sur papier
2.6 Checkbutton	4.3 Listbox	6.3 Fontes	
2.7 Radiobutton	4.4 Scrollbar	6.4 Couleurs	
2.8 Spinbox ^{Nv}	4.5 Menu	6.5 Curseurs souris	
2.9 Scale ^{Nv}			

1. Utilisation du module Tkinter

Tout d'abord, le module Tk pour python doit être installé. Pour Debian-Lenny, Squeeze ou Wheezy:

```
su apt-get install python-tk
```

Attention: lors de l'installation sous Debian Wheezy, il a fallu un reboot pour que les scripts ci-dessous fonctionnent.

Pour python3, le paquet nécessaire est `python3-tk`, du moins avec GNU/Linux Debian.

1.1 Charger le module ^{Rév. 2014.10}

Pour utiliser le **module Tkinter**, il faut d'abord choisir un mode d'importation.

import Tkinter charge le module, dont le nom doit précéder les méthodes de création d'objet et les constantes (explications au point suivant):

```
import Tkinter
racine0=Tkinter.Tk()
bouton0=Tkinter.Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack(side=Tkinter.RIGHT)
racine0.mainloop()
```

- **Tkinter.Tk()** ouvre l'objet fenêtre `racine0` duquel dépendront d'autres objets créés par la suite
- **Tkinter.Button()** crée l'objet `bouton0`
- **pack()** s'applique à un l'objet bouton représenté par la variable `bouton0`, qui ne s'affiche pas sans cela, voir [pack / grid](#)
- **Tkinter.RIGHT** est une constante utilisée pour l'**alignement**, qui peut être remplacée par "right" (sans préfixe)
- **mainloop()** ferme la boucle définie par **Tk()**, en s'appliquant sur l'objet `racine0`

Tk() et **Button()**, qui créent des objets, sont précédés du nom du module. Les fonctions créatrices d'objet prennent une capitale.

pack() et **mainloop()** sont des méthodes qui s'appliquent à des objets déjà créés; la programmation orientée objet est assez intelligente pour savoir qu'il s'agit d'objets créés par **Tkinter**: il est interdit de préfixer ces méthodes, qui s'écrivent sans capitale.

Alias

import Tkinter as tk permet d'utiliser l'alias `tk`: `racine0=tk.Tk()` remplace `racine0=Tkinter.Tk()`, etc. Les alias **T** ou **zzz** font tout aussi bien l'affaire.

```
import Tkinter as zzz
racine0=zzz.Tk()
bouton0=zzz.Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack(side=zzz.RIGHT)
racine0.mainloop()
```

Importation brute

from Tkinter import * permet d'utiliser toutes les fonctions de création d'objet et les constantes sans les préfixer: elle sont considérées au même niveau que les autres fonctions de python (ce qui peut provoquer des interférences!)

```
from Tkinter import *
racine0=Tk()
bouton0=Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack(side=LEFT)
racine0.mainloop()
```

Par souci de clarté, les exemples de ces pages importent toujours le module avec **import Tkinter**, ce qui rend le préfixage par **Tkinter** toujours nécessaire et explicite.


En python3, le module est rebaptisé `tkinter`. Il faudra alors choisir un de ces chargements de module:

```
import tkinter
import tkinter as Tkinter # pour la compatibilité avec cette page (sauf sous-modules, voir 8.)
import tkinter as tk      # pour un préfixage tk
from tkinter import *     # pour éviter tout préfixage
```

1.2 Ouvrir une fenêtre Rév. 2014.10

L'utilisation d'une fenêtre se résume schématiquement à cette procédure (les majuscules sont toujours importantes):

```
import Tkinter # pour python3: import tkinter as Tkinter
racine0=Tkinter.Tk()
racine0.title("Un titre arbitraire")
racine0.geometry("200x150+100+50")
racine0.aspect(3, 2, 5, 3)
bouton0=Tkinter.Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack(side=Tkinter.RIGHT)
racine0.mainloop()
```

- **Tkinter.Tk()** ouvre l'objet racine, le «parent» le plus élevé. Il s'agit d'un premier objet (une fenêtre redimensionnable) duquel les autres dépendront
- **racine0.title()** détermine le titre de la fenêtre, qui s'inscrit dans la barre du haut
- **geometry("largeurxhauteur+Δ largeur+Δ hauteur")** précise les dimensions minimales intérieure de la fenêtre (**x** minuscule entre les deux dimensions exprimées en pixels), ainsi que l'écartement à gauche et en haut par rapport à l'écran. Attention si votre système comporte une barre supérieure, qui ne sera pas recouverte. Gnome2/Mate-desktop en tout cas repousse la fenêtre vers le bas de la barre supérieure en cas d'écart insuffisant.
- **fenetre0.aspect(x1, y1, x2, y2)** impose, pour un redimensionnement (par exemple à la souris), un rapport entre x1/y1 et x2/y2.
 - d'après exérimentation, il semble que x1/y1 doive être plus petit que x2/y2
 - ces rapports doivent être compatibles avec la largeur et la hauteur de **geometry**, sinon la fenêtre est basée sur la largeur de **geometry**, la hauteur étant limitée à la valeur la plus proche déterminée par x1/y1 et x2/y2
- **Tkinter.Button()** ajoute le widget «bouton à cliquer». C'est un objet «enfant», qui doit préciser son parent direct comme premier paramètre
- **pack()** est avec **grid** la méthode qui installe le widget précisé dans son widget-parent
- **Tkinter.RIGHT** est la constante "**right**", qui **align**e le widget à droite, ce qui une façon d'organiser les widgets entre eux
- **mainloop()** est une méthode toute spéciale, qui attend les événements (commandes liées aux boutons et renvoyant à des fonctions, temporalité...). On en sort le plus souvent avec **command=racine0.quit** associé au clic d'un bouton comme dans l'exemple ci-dessus, ou par la fermeture par un clic sur l'icone  de la barre supérieure de la fenêtre créée avec **Tkinter.Tk()**

TopLevel

TopLevel permet l'ouverture d'une nouvelle fenêtre. Afin de ne pas surcharger l'exemple, la nouvelle fenêtre n'affiche rien de plus que la première.

```
import Tkinter

def top():
    fenetre0=Tkinter.Toplevel()
    fenetre0.title("Seconde")

racine0=Tkinter.Tk()
racine0.title("Principale")
bouton0=Tkinter.Button(text="Autre", command=top)
bouton0.pack()
racine0.mainloop()
```

- le bouton [Autre] active la fonction **top()**, qui ouvre la seconde fenêtre
- cette nouvelle fenêtre ne requiert ni **pack()** ni **mainloop()**
- fermer la principale ferme la seconde (sortie de **mainloop()**), l'inverse n'est pas vrai.

Attention! les variables représentant les widgets définis dans une fonction ne sont pas globales! Il faudra placer une ligne **global** suivie de ces variables si on veut les utiliser à d'autres endroits du script.

1.3 Lancer le script

Tous les exemples proposés sur cette page sont fonctionnels. Il y a plusieurs méthodes pour **lancer** un script écrit en python.

Sur un système UNIX, copier-coller chaque exemple dans un fichier-texte, nommé par exemple **exemple.py**. Ouvrir une console, naviguer vers le répertoire contenant le fichier (ici: **scripts-py**) et le lancer avec

```
cd scripts-py
python exemple.py
```

ou le lancer avec son chemin complet (chemin selon votre nom d'utilisateur)

```
python /home/dudule/scripts-py/exemple.py
```

Il est possible d'appeler implicitement python (en Unix) en spécifiant sur la première ligne (**-Qnew** sert à obtenir la division «réelle», inutile si l'on utilise python3):

```
#!/usr/bin/python -Qnew
```

Il faut ensuite rendre le fichier exécutable avec **chmod 740 nomdufichier.py** et le lancer avec **./exemple.py** ou **chemin/exemple.py**

Par l'interface graphique

Voir la section **Lancement** du manuel.

Caractères non ASCII

Par ailleurs, pour disposer des accents dans les chaînes et les commentaires d'un script, il faut placer au plus tôt (après un éventuel **#!/usr/bin/python**, voir **lancer**):

```
# -*- coding:utf-8 -*-
```

ou tout autre codage que vous utilisez pour la console: `coding:latin-1` (ou `iso8859-1`), `coding:latin-15` (ou `iso8859-15`)...

2 Les widgets simples

«widget» est paraît-il une contraction de «window gadget». Il s'agit d'éléments simples, comme un titre, un bouton, un texte, un champ éditable... ou plus complexes, comme un système de **menu**. Lors de la mise en place d'un widget, la récupération de son identifiant dans une variable permet par la suite de le contrôler.

2.1 Frame et LabelFrame

Frame est un cadre, permettant de regrouper géographiquement les widgets dans une fenêtre.

```
import Tkinter

racine0=Tkinter.Tk()
racine0.geometry("400x300")
cadre0=Tkinter.Frame(racine0)

bouton1=Tkinter.Button(cadre0, text="Bouton 1")
bouton1.pack(side=Tkinter.LEFT)
bouton2=Tkinter.Button(cadre0, text="Bouton 2")
bouton2.pack(side=Tkinter.TOP)
bouton3=Tkinter.Button(cadre0, text="Bouton 3")
bouton3.pack()

cadre0.pack()

racine0.mainloop()
```

- La disposition des boutons changera selon l'ordre et l'orientation qu'on leur donne.
- Un cadre est disposé par défaut en haut de la fenêtre. Il est possible de modifier cela avec `pack(side=)`
- Un cadre resserre autant que possible ses composants. Il est possible de lui ajouter une marge à gauche et à droite avec `padx=` et en haut et en bas avec `pady=` (en pixels) dans `pack()`

```
Tkinter.Frame(parent, bg="red", border=3, relief=Tkinter.GROOVE)
```

- `bg=""` permet une définition de **couleur** de fond
- `border=` décide de la largeur en pixels de la ligne de contour
- `relief=""` permet de définir un type de **relief**

Il est possible de donner un bord et un nom à un cadre avec LabelFrame:

```
cadre0=Tkinter.LabelFrame(parent, text="")
```

- `text=""` détermine le titre du cadre

2.2 Label

Label permet un affichage simple de texte:

```
import Tkinter
racine0=Tkinter.Tk()
mot0=Tkinter.Label(racine0, text="Premier texte\ndans une fenetre")
mot0.pack(side=Tkinter.BOTTOM)
racine0.mainloop()
```

- La variable `mot0`, qui contient l'identificateur du widget, permet de l'installer avec `mot0.pack()` et éventuellement le modifier ou le supprimer [lien interne]
- `side=Tkinter.BOTTOM` garde le texte au bas de la fenêtre si on l'agrandit.

Label peut également recevoir une image sous format **GIF/PNM** ou **bitmap**:

```
import Tkinter
racine0=Tkinter.Tk()
dessin0=Tkinter.PhotoImage(file="nomdefichier.gif")
etiquette0=Tkinter.Label(image=dessin0)
etiquette0.pack()
racine0.mainloop()
```

- `file` désigne un nom de fichier, éventuellement avec un chemin suffisant

2.3 Texte éditable

Text définit une plage permettant l'insertion et la manipulation d'un texte.

```
import Tkinter
racine0=Tkinter.Tk()
texte0=Tkinter.Text(racine0, width=25, height=5)
texte0.insert("end", "bla bla bla")
texte0.pack(side=Tkinter.RIGHT)
racine0.mainloop()
```

- `width` est la largeur de la plage en nombre de caractères, `height` le nombre de lignes
- `side=Tkinter.RIGHT` dans `pack()` concerne l'**alignement** de la plage de texte: en agrandissant la fenêtre, on verra que la plage de texte se situe à droite, même si le texte est aligné à gauche.
- `insert(Tkinter.END, chaine)` permet l'insertion d'une chaîne à la fin du texte, `Tkinter.INSERT` (par défaut) à l'endroit du curseur-texte, `CURRENT` à l'endroit le plus proche du curseur souris?
- `state=Tkinter.DISABLED` interdit l'édition du texte, `state=Tkinter.NORMAL` (par défaut) la permet, mais cela ne peut être précisé qu'après l'insertion du texte avec `texte0.config(state=Tkinter.DISABLED)`

`tag_config()` permet de préciser un style à des portions de texte préalablement définies avec `add_tag()`

```
import Tkinter

racine0=Tkinter.Tk()

texte0=Tkinter.Text(racine0)
texte0.config(font="sans 12", width=17, height=4)
texte0.pack()
texte0.insert(Tkinter.END, "Les sanglots longs\nDes violons\nDe l'automne")

texte0.tag_add("accent", '1.13', '1.18')
texte0.tag_add("accent", '2.7', '2.11')
texte0.tag_add("accent", '3.7', '3.10')
texte0.tag_config("accent", font="monospace 12 underline", foreground="#aa0055")

racine0.mainloop()
```

- **texte0.tag_add()** définit des plages de texte pour une configuration précise, relié à un «tag». Dans notre exemple, première ligne: les caractères de 13 à 18 (0 est le premier); pour la seconde ligne, les caractères de 7 à 11 (\n séparent deux lignes)...
- **texte0.tag_config()** applique une configuration aux plages de textes taggués.
- **foreground=** ne peut être abrégé en **fg=**

2.4 Entry Rév. 2014.10

Entry crée un champ permettant de saisir une chaîne ou un nombre, entier ou «réel». Il faut donc prévoir une variable permettant de recevoir le texte saisi, que l'on peut définir par défaut avec **texte0.set()**.

```
import Tkinter
racine0=Tkinter.Tk()
invite0=Tkinter.Label(racine0, text='Cliquer et saisir:', width=20, height=3, fg="navy")
invite0.pack()
texte0=Tkinter.StringVar() # definition d'une variable-chaîne pour recevoir la saisie d'un texte
texte0.set("Sans commentaire") # facultatif: assigne une valeur à la variable
saisie0=Tkinter.Entry(textvariable=texte0, width=30)
saisie0.pack()
racine0.mainloop()
print texte0.get() # affiche le texte saisi à la fermeture de la fenêtre
```

- **entier0=Tkinter.IntVar()** permet de saisir une valeur entière, qui n'accepte que les chiffres de 0 à 9, + et -
- **reel0=Tkinter.DoubleVar()** permet de saisir une valeur «réelle», acceptant également le point décimal ., e ou E pour la notation scientifique de type **43e23** (43×10^{23}) ou **43e-23** ($43/10^{23}$). Si la valeur saisie est entière, la variable se terminera par .0 pour garder son type «réel».

2.5 Button Rév. 2014.10

Button définit un bouton cliquable

```
import Tkinter
racine0=Tkinter.Tk()
bouton0=Tkinter.Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack(side=Tkinter.BOTTOM)
racine0.mainloop()
```

- **width=** et **height=** permettent de déterminer la largeur et la hauteur du bouton en hauteur de caractères, ou en pixels en cas d'utilisation d'une image.
- **relief=** permet de déterminer le style de **relief** du bouton. Par défaut c'est **relief=Tkinter.RAISED**, avec une animation vers **Tkinter.SUNKEN** lorsqu'il est cliqué.
- l'**alignement** de **pack()** avec **side=** peut également prendre les valeurs **Tkinter.LEFT**, **Tkinter.RIGHT**, **Tkinter.TOP** ou **Tkinter.CENTER**

Il est à noter que le nom de la fonction-cible de **Tkinter.Button** s'écrit sans guillemet ni parenthèse. Des parenthèses (par exemple pour envoyer un paramètre à la fonction) ne sont pas utilisables parce qu'elles ont pour effet de lancer la commande au chargement du script. S'il y a peu de paramètres (par exemple trois boutons différents qui envoient chacun une valeur, il est possible de définir chaque bouton avec **command=aide0**, **command=aide1**... et les fonctions correspondantes: **def aide0: aide(0)**, **def aide0: aide(1)**... renvoyant à **def aide(n):**. Une autre manière serait d'utiliser une variable globale, récupérée avec **global** à l'intérieur de la fonction appelée.

bitmap

Il existe une série d'images toutes faites (pour les nostalgiques des premières icônes N/B) pour ces boutons: **bitmap="gray12"**, "gray25", "gray50", "gray75", "warning", "hourglass", "info", "questhead", "question", "error", à utiliser de la sorte:



```
import Tkinter
racine0=Tkinter.Tk()
bouton0=Tkinter.Button(racine0, text="Attention", bitmap="error", compound=Tkinter.RIGHT)
bouton0.pack()
racine0.mainloop()
```

compound=Tkinter.TOP, **Tkinter.BOTTOM**, **Tkinter.LEFT**, **Tkinter.RIGHT**, indique la position de l'icône par rapport au texte, et peut également prendre la valeur **"center"** (superposition): voir également **Alignements**.

Il est - fort heureusement! - possible d'utiliser une **image** personnelle initialisée avec **BitmapImage()** et spécifiée avec **image=**.

2.6 Checkbutton

Checkbutton est une "case à cocher". Une méthode est proposée pour récupérer l'information sur l'état de la "case à cocher".

```
import Tkinter
racine0=Tkinter.Tk()
retour0=Tkinter.IntVar() # creation de variable-retour
bouton0=Tkinter.Checkbutton(racine0, variable=retour0, text="Cochez-moi")
```

```
bouton0.pack()
racine0.mainloop()

# recuperation de la valeur lors de la sortie de la boucle mainloop():
if retour0.get(): # la variable 'retour0' = 1 si la case est cochée, 0 sinon
    print "Tilt!"
else:
    print "Vide!"
```

Pour cocher (ou décocher) un `CheckBox` par voie de script, `retour0.set(1)` positionne la valeur à 1 (ou à 0). Doit être suivi de `bouton0=Tkinter.Checkbutton` ou de `bouton0.config()`

- `indicatoron=0` est un paramètre qui permet de transformer une case à cocher en bouton «off» (relevé, =0, texte sur fond gris) / «on» (enfoncé, =1, texte sur fond blanc). Pour apprécier le relief, il vaut mieux prévoir une bordure suffisante, à partir de `border=2`.

2.7 Radiobutton

Les `RadioButton` sont un ensemble de `CheckBox` reliés par la même variable, et dont un n'est sélectionné qu'à l'exclusion des autres. Cette variable-retour rendra la valeur proposée par `value` qui aura été sélectionné.

```
import Tkinter
racine0=Tkinter.Tk()
retour0=Tkinter.IntVar() # cree une variable entiere pour recevoir la valeur retour
retour0.set(2) # le bouton [Bof] mis par default (value=2)
bouton1=Tkinter.Radiobutton(racine0, text="Oui", variable=retour0, value=1, bd=2)
bouton2=Tkinter.Radiobutton(racine0, text="Bof", variable=retour0, value=2, bd=3)
bouton3=Tkinter.Radiobutton(racine0, text="Non", variable=retour0, value=3, bd=3)
bouton1.grid(row=0, column=0)
bouton2.grid(row=0, column=1)
bouton3.grid(row=0, column=2)
racine0.mainloop()

print retour0.get() # retourne 1, 2 ou 3 selon le bouton choisi, ou 0 si pas de choix
```

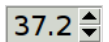
- `retour0=Tkinter.StringVar()` permet d'initialiser une variable pour une valeur de retour sous forme de chaîne. `value` doit alors recevoir une «chaîne»
- `retour0.set()` permet de forcer une valeur par défaut, et donc un bouton enfoncé. Ce n'est pas obligatoire, mais un bouton apparaîtra coché si sa valeur de retour est 0
- `retour0=Tkinter.IntVar()` initialise `retour0` comme variable entière; `Tkinter.DoubleVar()` pour une variable «réelle» et `Tkinter.StringVar()` pour une variable-chaîne
- `variable=retour0` regroupe les boutons d'un même groupe de radio-boutons: tous ceux qui jouent ensemble doivent avoir la même variable-retour.
- `value=` différencie chaque bouton d'un groupe de radio-boutons, la valeur est rendue par `retour0.get()`
- `grid()` est un équivalent de `pack()` permettant le placement des widget dans un système de rangées/colonnes

Et encore...

- `indicatoron=0` permet de transformer une case à cocher en bouton «off» (relevé, =0, texte sur fond gris) / «on» (enfoncé, =1, texte sur fond blanc). Une bordure à partir de `border=2` permet de mieux apprécier le relief.

2.8 Spinbox Nv. 2014.10

Widget permettant de choisir entre plusieurs valeurs que l'on fait défiler en cliquant sur des flèches. Les valeurs peuvent être des nombres, entiers ou réels, ou des chaînes, définis soit entre deux bornes (nombres) ou dans des objets itérables (nombres ou chaînes), ou des fonctions les produisant.



```
import Tkinter
racine0=Tkinter.Tk()
retour0=Tkinter.StringVar()
retour0.set(37.2)
spin0=Tkinter.Spinbox(racine0, from_=35, to=43, increment=.2, width=4)
spin0.config(textvariable=retour0, font="sans 24", justify="center")
spin0.pack()
racine0.mainloop()

print retour0.get()
```

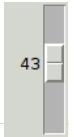
- `retour0=Tkinter.StringVar()` initialise la variable-retour `retour0`, qui sera ensuite récupérée par `retour0.get()` sous forme de chaîne même à partir de valeurs entières ou «réelles».
- `retour0.set()` précise la valeur de la variable-retour avant la définition du widget
- `textvariable=retour0` précise la variable-retour dans le widget
- `from_`, `to`, `increment` définissent respectivement un début, une fin et une éventuelle incrémentation, qui peuvent être entiers ou «réels». Contrairement à `to`, le début (`from_`) n'est pas obligatoire, valant 0 par défaut. Des valeurs négatives sont possibles, tant que `to` est plus grand que `from_`; `increment` doit être strictement positif. `from_` doit son tiret bas au fait que `from` est un nom réservé en python.
- `font="sans 24"` permet d'agrandir la fonte, et de ce fait les flèches à cliquer
- `width` précise la largeur du texte en nombre de caractères (20 par défaut)
- `spin0.config()` permet d'écrire les paramètres en plusieurs fois

Et encore...

- `values=` alternative à `from_` / `to`, ce paramètre peut recevoir une liste `[1, "qsd", "7", 8]`, un tuple `(3, "A", 45)` ou une fonction renvoyant un objet itérable, comme `range()`
- `buttonbackground` définit la couleur des flèches
- `repeatdelay`= nombre de millisecondes d'appui sur une flèche avant que la valeur ne commence à changer automatiquement (400 par défaut, semble un peu lent)
- `repeatinterval`= nombre de millisecondes entre deux changements de valeur (100 par défaut, semble un peu rapide)
- ...

2.9 Scale Nv. 2014.10

Le widget **scale** permet de déterminer à la souris une valeur numérique entre deux bornes. Le curseur se manipule par saisie/déplacement (*drag and drop*) ou en cliquant sur la coulisse en deçà ou au-delà du curseur pour un déplacement plus fin. Avis aux distraits: la figure ci-contre n'en est qu'une image figée, pas le widget lui-même.



L'exemple suivant, très simple, définit un curseur vertical balayant les valeurs de 0 à 100 sur... 101 pixels.

```
import Tkinter
racine0=Tkinter.Tk()
retour0=Tkinter.IntVar()
retour0.set(43)
echelle0=Tkinter.Scale(racine0, variable=retour0)
echelle0.pack()
racine0.mainloop()

print retour0.get()
```

- **retour0=Tkinter.IntVar()** initialise une variable-retour entière
- **retour0.set(43)** fixe la variable-retour à 43
- **variable=retour0** assigne la variable retour au widget
- **print retour0.get()** affiche la variable retour lors de la sortie de mainloop()

Paramètres additionnels

- **from_**, **to**= déterminent les valeurs basse et haute (incluses) entre lesquelles le curseur peut coulisser (0 et 100 par défaut).
- **resolution**= détermine l'écart entre deux valeurs (1 par défaut). C'est de cet écart que le curseur bouge lors d'un clic sur la coulisse.
- **length**= définit la longueur de l'échelle (101 pixels par défaut, plus l'épaisseur du curseur).
 - s'il y a plus de valeurs que de pixels, certaines valeurs seront manquantes par saisie/déplacement du curseur: il faut dans ce cas cliquer en deçà ou au delà du curseur pour accroître ou diminuer la valeur d'une unité
 - s'il y a moins de valeurs que de pixels, le curseur saute aux valeurs justes, sans s'arrêter aux valeurs intermédiaires
- **width**= largeur en pixels du curseur et de la coulisse (15 par défaut)
- **sliderlength**= épaisseur du curseur (30 par défaut)
- **orient**= permet de définir le sens du curseur: "horizontal" (accroissement vers la droite) ou "vertical" (accroissement vers le haut, par défaut)
- **repeatdelay**= nombre de millisecondes d'appui en deçà ou au delà du curseur avant que la valeur ne commence à changer automatiquement (400 par défaut, semble un peu lent)
- **repeatinterval**= nombre de millisecondes entre deux changements de valeur (100 par défaut, semble un peu rapide)
- **command**= détermine une fonction/procédure lors de la sollicitation du widget. Attention: la fonction définie par **def valeur()**: est appelée par **command=valeur**. Les variables définies par Tkinter.IntVar() sont globales.
- **showvalue=0** supprime l'étiquette qui indique la valeur courante, à utiliser avec le paramètre **command**= qui permet d'afficher la valeur ailleurs dans le cadre.

avec valeurs réelles

Il est possible de déterminer un **scale** avec valeurs réelles.

```
import Tkinter
racine0=Tkinter.Tk()
retour0=Tkinter.DoubleVar()
retour0.set(18.15)
echelle0=Tkinter.Scale(racine0, variable=retour0, length=300, resolution=.05)
echelle0.pack()
racine0.mainloop()
print retour0.get()
```

- **retour0=Tkinter.DoubleVar()** initialise nécessairement une variable-retour «réelle»
- **resolution**= détermine un écart différent de l'unité, par exemple .05. C'est de cet écart que le curseur bouge lors d'un clic sur la coulisse.

3. L'espace graphique Canvas

Canvas crée une surface sur laquelle on peut placer des éléments graphiques (l'exemple suivant n'est pas fonctionnel par manque de commandes):

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=300, height=200, background='darkgray')
fond0.pack()
# ... votre script
racine0.mainloop()
```

Pour la plupart des éléments graphiques présentés ci-dessous, **fill** définit la couleur (le texte pour **creat_text**), **width** l'épaisseur, **outline** la couleur du bord, **anchor** l'alignement.

3.1 Texte graphique

Pour positionner un texte au pixel près sur un canevas (attention: le positionnement par défaut est le centre du texte, comme le montrent les deux lignes)

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=150, height=120, background='darkgray')
ligne1=fond0.create_line(75, 0, 75, 120)
ligne2=fond0.create_line(0, 60, 150, 60)
texte0=fond0.create_text(75, 60, text="Spam?", font="Arial 16 italic", fill="green")

fond0.pack()
racine0.mainloop()
```

Rappel: la couleur du texte est définie ici par **fill**!

3.2 Lignes et points

Avec **fond0** représentant l'ouverture d'un canevas:


```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=350, height=200, background='darkgray')
fond0.pack()
ligne=fond0.create_line(40, 190, 250, 110, 270, 170, 180, 120)
racine0.mainloop()
```

Les arguments commencent par une série de paires qui sont des coordonnées de points (deux paires minimum pour un segment de droite).

Styles de ligne

- **width=** nombre de pixels pour épaissir le trait

Il est possible de déterminer un style pour les extrémités de ligne (si l'épaisseur est suffisante):

- **capstyle=Tkinter.ROUND** fin de ligne arrondie
- **capstyle=Tkinter.BUTT** fin de ligne coupée
- **capstyle=Tkinter.PROJECTING** fin de ligne coupée, une demi-largeur au delà des coordonnées du point
- **arrow=Tkinter.BOTH** place une flèche aux bouts de la ligne (Tkinter.FIRST pour le départ, Tkinter.LAST pour la fin)
- **arrowshape=(8, 10, 3)** permet de modifier la forme de la flèche - (10, 20, 8) pour un losange
 - longueur de la partie centrale (plus courte que la valeur suivante pour une flèche en ancre)
 - longueur des ailes à partir de la pointe
 - largeur de la flèche à l'extrémité des ailes

Les angles de lignes brisées peuvent également recevoir un style:

- **joinstyle=Tkinter.ROUND** angle de deux segments arrondi
- **joinstyle=Tkinter.MITER** angle de deux segments pointu
- **joinstyle=Tkinter.BEVEL** angle de deux segments coupé (orthogonale de la bissectrice)

Il est possible de donner un style à la ligne

Lignes discontinues Aj. 2015.01

- **dash=(n, q)** force la ligne brisée avec *n* pixels en couleur suivis de *q* pixels d'interruption. Il est possible de définir des lignes complexes avec un tuple plus long (n0, q0, n1, q1...)
- **dashoff=d** définit le début de la séquence «dash»

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=200, height=100, background='darkgray')
ligne=fond0.create_line(30, 50, 170, 50, dash=(7, 2, 2, 2), dashoff=2)
fond0.pack()
racine0.mainloop()
```

Courbe de Bézier (spline)

smooth=True lisse une ligne brisée (spline). Attention: si la courbe commence et termine sur les points extrêmes, elle ne passe pas par les points intermédiaires.

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=400, height=400)
fond0.pack()
ligne1=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True, splinesteps="1", fill="orange")
ligne2=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True, splinesteps="2", fill="green")
ligne3=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True, splinesteps="3", fill="purple")
ligne5=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True, splinesteps="5", fill="blue")
ligne9=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True, splinesteps="9", fill="red")
ligne43=fond0.create_line(10, 10, 10, 380, 380, 380, 380, 10, smooth=True)
racine0.mainloop()
```

- **splinesteps=** définit un degré de lissage (1 pour moins de brisure que le nombre de traits, 12 approche du lissage maximal).

Dessiner un point

Pour afficher un point x, y, il faut afficher une ligne qui va de x, y à x+1, y ou x, y+1 (le dernier point d'une ligne n'est pas affiché):

```
import Tkinter
racine0=Tkinter.Tk()
racine0.title("Regardez au centre!")
fond0=Tkinter.Canvas(racine0, width=350, height=200, background='darkgray')
ligne0=fond0.create_line(100, 100, 101, 100)
fond0.pack()
racine0.mainloop()
```

3.3 Surfaces Rév. 2014.12

Pour **fond0** représentant l'ouverture d'un canevas (les couleurs ont été exécutées pour l'exemple):

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=350, height=200, background='darkgray')
rectangle0=fond0.create_rectangle(50, 40, 300, 90)
ellipse0=fond0.create_oval(30, 120, 150, 180)
quartier0=fond0.create_arc(160, 130, 230, 200, start=30, extent=120, style=Tkinter.PIESLICE)
arc0=fond0.create_arc(250, 130, 320, 200, start=30, extent=120, style=Tkinter.CHORD)
fond0.pack()
racine0.mainloop()
```

- **create_rectangle:** les deux premiers entiers représentent les coordonnées du point en haut à gauche du rectangle, les deux suivants celles du point en bas à droite
- **create_oval:** il s'agit des coordonnées du rectangle circonscrit à l'ovale (pour un cercle, largeur et hauteur doivent être égales)

- **create_arc**: il s'agit des coordonnées du rectangle circonscrit à l'ovale entier, quelques soient les dimensions de la portion
- **create_arc**: **start**= définit l'angle de départ, **extent**= l'angle d'arrivée, exprimés en degrés (sens trigonométrique, «réels» permis)
 - **style=Tkinter.PIESLICE** dessine un quartier de tarte (par défaut)
 - **style=Tkinter.CHORD** dessine l'arc et sa corde.
- **fill**= définit la couleur de remplissage de la forme géométrique (transparente par défaut)
- **color**= définit la couleur de contour de la forme géométrique (noire par défaut)
- **width**= définit l'épaisseur du contour de la forme géométrique (1 par défaut)

Notes Aj. 2015.01

Contrairement au point terminal d'une ligne qui n'est pas dessiné, ces formes géométriques sont dessinées jusqu'aux coordonnées extrêmes.

Comme pour les **lignes**, **dash** et **dashoff** sont disponibles pour le contour des formes géométriques, mais la version Tkinter 2.7.3 ajoute un pixel visible (1 vaut 2, 0 n'est pas permis: un pixel unique n'est donc pas définissable) et enlève un pixel d'interruption (1 est réduit à 0, ce qui donne une ligne continue): **dash=(2, 2)** définit la ligne **xxx xxx xxx** !

```
import Tkinter

racine0=Tkinter.Tk()

fond0=Tkinter.Canvas(racine0, width=200, height=100, background='darkgray')
line0=fond0.create_line(30, 20, 170, 20, dash=(7, 1, 1, 1))
rectangle0=fond0.create_rectangle(30, 30, 170, 50, dash=(7, 2, 1, 2))
ellipse0=fond0.create_oval(30, 60, 170, 80, dash=(7, 2, 1, 2))
fond0.pack()

racine0.mainloop()
```

Polygones

Pour dessiner un polygone, éventuellement arrondi, il faut en déterminer les différentes coordonnées de points (sans se préoccuper de répéter les coordonnées du premier):

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, width=150, height=150, background='darkgray')
polygone0=fond0.create_polygon(35, 105, 120, 85, 95, 25, 80, 75, 25, 60, 65, 30, fill="cyan", width=5, outline='black')
fond0.pack()
racine0.mainloop()
```

- **smooth=True** et **splinessteps=n** sont disponibles pour les polygones.

3.4 Afficher un fichier-image Rév. 2015.04

Tkinter ne reconnaît que les formats GIF et certains PNM (voir plus **bas**). L'affichage se fait en deux étapes: d'abord **PhotoImage()** qui charge un fichier avec **file=""** ou qui précise la chaîne de données avec **data=**, et puis l'affichage de l'image avec **create_image()**, dans un canevas préalablement défini.

Pour une image existante sur le disque dur, on utilise donc **file="votre_image"** (avec un chemin d'accès si l'image est dans un autre répertoire).

```
import Tkinter
racine0=Tkinter.Tk()
photo0=Tkinter.PhotoImage(file="exemple.gif") # ouverture du fichier GIF
largeur=photo0.width(); hauteur=photo0.height() # determination des dimensions
racine0.geometry(str(largeur+2)+"x"+str(hauteur+2))
fond0=Tkinter.Canvas(racine0, bg='gray')
fond0.pack()
image0=fond0.create_image(largeur//2+1, hauteur//2+1, image=photo0) # image a centrer
racine0.mainloop()
```

- pour cet exemple, le fichier **exemple.gif** doit exister dans le répertoire du script
- **photo0** est la variable créée lors du chargement de l'image, elle permet par exemple de déterminer les dimensions de l'image avec **photo0.width** et **photo0.height**, utiles pour définir les dimensions du canevas.
- **create_image()** affiche l'image sur le canevas représenté par la variable **fond0**
- **image0** est la variable créée lors du placement de l'image sur le canevas, et permet de **reconfigurer ou supprimer** l'image
- **image0=fond0.create_image(0, 0, image=photo0, anchor=Tkinter.NW)** aligne l'image par rapport en haut et à gauche du canevas (défini par **fond0**) ; les argument sans paramètres (abscisse, ordonnée, largeur, hauteur...) doivent être donnés avant les argument avec paramètre

Couleur d'un pixel d'une image

Il est possible d'interroger la valeur d'un pixel d'une image ou d'en modifier sa couleur. **get()** et **put()** s'appliquent sur la variable générée par l'ouverture du fichier **PhotoImage()**

- **couleur=photo0.get(xx, yy)** retourne un tuple des valeurs de rouge, vert et bleu du point xx, yy (0, 0 est en haut à gauche)
- **photo0.put("#234a87", (xx, yy))** colore le pixel (xx, yy) de l'image ouverte avec **photo0** de la nuance **#234a87**.

2015.03 **Note:** Tkinter ne gère pas les .GIF animées. Pour une animation, il convient d'utiliser plusieurs .GIF, d'utiliser le positionnement, la définition et la suppression d'images, contenant éventuellement de la transparence. Typiquement, les images sont chargées une fois et le canevas est défini une fois; à chaque changement (**create_image** et **delete**), il faut réutiliser **pack()** appliqué sur canevas, et temporaliser les modifications avec **after()**.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import Tkinter, math

i=0
# definition des images
soleil="P5 7 7 255 "+chr(255)*3+chr(0)+chr(255)*3+(chr(255)+chr(0))*3+chr(255)+chr(255)*2+chr(0)+chr(255)
soleil+=chr(0)+chr(255)*2+chr(0)*2+chr(255)*3+chr(0)*2+chr(255)*2+chr(0)+chr(255)+chr(0)+chr(255)*2
```



```
soleil+=(chr(255)+chr(0))*3+chr(255)+chr(255)*3+chr(0)+chr(255)*3
terre="P5 5 5 255 "+chr(255)+chr(0)*3+chr(255)+(chr(0)+chr(255)*3+chr(0))*3+chr(255)+chr(0)*3+chr(255)

def temps():
    global i
    i+=1;angle=i*math.pi/180
    fond0.coords(imagel, (100-70*math.sin(angle), 100-math.cos(angle)*30))
    tps0=racine0.after(10, temps)

racine0=Tkinter.Tk()
```

Formats PNM: PGM et PPM Rév. 2014.12

Seuls les PNM au format brut (binaire) en nuances de gris (PNM ou PGM) ou couleurs (PNM ou PPM), à savoir ceux dont l'entête commence par P5 ou P6, peuvent être affichés par Tkinter. Pour des renseignements sur ces formats non compressés et faciles à générer: **P5** et **P6**.

data= permet de définir une chaîne contenant les données d'une image que vous produisez vous-même:

```
import Tkinter
p5="P5 8 8 255 "+((chr(255)*2+chr(0)*2)*4+(chr(0)*2+chr(255)*2)*4)*2
# (2px blancs suivis de 2px noirs)x4 + (2px noirs suivis de 2px blancs)x4,
# le tout x2, fait un damier de 4x4 carrés de 2x2px
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0)
damier0=Tkinter.PhotoImage(data=p5)
imagel=fond0.create_image(120, 120, image=damier0)
fond0.pack()
racine0.mainloop()
```

Vous trouverez sur la page des **recettes** une façon de créer une image d'un cercle à partir d'un rayon quelconque.

3.5 Afficher une image bitmap Rév. 2014.10

Tkinter permet d'utiliser une image bitmap bicolore, par exemple en exportant une image noir et blanc avec l'extension **.xbm** avec Gimp. Chaque pixel noir du dessin correspond à un bit mis à 1 (couleur définie par **foreground**), et chaque pixel blanc correspond à un bit mis à 0, qui laisse la couleur de fond du canevas inchangée, ou celle définie par **background** de **BitmapImage()**.

```
import Tkinter
dessin0=""
# define im_width 24
# define im_height 16
# static char im_data[] = {
0x0f, 0x0, 0xcc, 0x0f, 0x0, 0xcc, 0x0f, 0x0, 0x33, 0x0f, 0x0, 0x33, 0xf0, 0x0, 0xcc, 0xf0,
0x0, 0xcc, 0xf0, 0x0, 0x33, 0xf0, 0x0, 0x33, 0x0, 0xff, 0x0, 0x0, 0xff, 0x0, 0x0, 0xc3,
0x0, 0x0, 0xc3, 0x0, 0x0, 0xc3, 0x0, 0x0, 0xc3, 0x0, 0x0, 0xff, 0x0, 0x0, 0xff, 0x0
};
"""
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, background='gray')
damier0=Tkinter.BitmapImage(data=dessin0, foreground="red", background="blue")
print damier0.width(), damier0.height()
image0=fond0.create_image(20, 20, image=damier0)
fond0.pack()
racine0.mainloop()
```

- L'encodage de l'image provient de la syntaxe du langage C, qu'il convient plus ou moins de respecter. Après expérimentation, les trois premières lignes doivent commencer par le **#** et respectivement comporter **_width** et une largeur en pixels, **_height** et une hauteur en pixels, et **char** suivi d'un nom de variable-liste dont le choix semble arbitraire.
- Chaque octet (ici sous forme hexadécimale) contient huit pixels, la séquence **10101010** étant codée **0xaa**, selon la conversion par quartet:

0000 → 0	0010 → 2	0100 → 4	0110 → 6	1000 → 8	1010 → a	1100 → c	1110 → e
0001 → 1	0011 → 3	0101 → 5	0111 → 7	1001 → 9	1011 → b	1101 → d	1111 → f

- Les octets définissent les paquet horizontal de huit pixels de gauche à droite et de haut en bas, mais chaque octet est inversé par rapport à la suite de pixels. Un ensemble de 8 pixels **.X.XXX..** se code en fait **00111010**, à savoir **x03A**

Il est possible de charger un fichier, au même format, avec **damier0=Tkinter.BitmapImage(file="image.xbm")**

Comme pour les fichiers-images, **damier0.width()** et **damier0.height()** retourne la largeur et la hauteur de l'image bitmap, pour la variable **damier0** issu de **Tkinter.BitmapImage()**.

Si **background** a été précisé, il est possible de définir un masque, au même format, qui décide des pixels affichés:

```
masque0=""
# define mask_width 24
# define mask_height 16
# static char mask[] = {
0xaa, 0xaa, 0xaa, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55,
0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55, 0x55, 0x55, 0xaa, 0xaa,
0xaa, 0x55, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0x55, 0x55, 0x55
};
"""
```

et l'appel doit être complété:

```
damier0=Tkinter.BitmapImage(data=dessin0, maskdata=masque0, foreground="black", background="white")
```

Il est possible de définir une image par un fichier reprenant le codage vu plus haut, avec **file=""** et **maskfile=""** (les adresses utilisées devraient être valides sur un système UNIX/X11):

```
import Tkinter
racine0=Tkinter.Tk()
fond0=Tkinter.Canvas(racine0, bg='gray')
```

```
dessin0="/usr/include/X11/bitmaps/star"
masque0="/usr/include/X11/bitmaps/starMask"
definition0=Tkinter.BitmapImage(file=dessin0, maskfile=masque0, foreground="red", background="white")
image0=fond0.create_image(20, 20, image=definition0)
fond0.pack()
racine0.mainloop()
```

- **maskfile** n'est pas obligatoire; il s'agit d'un tapis ajouré qui recouvre la couleur du fond à certains endroits:



```
* * * dessin (rouge)
--- --- --- masque (blanc)
===== fond (gris)
```

- **foreground** et **background** de **BitmapImage** ne peuvent être remplacés par **fg** et **bg**

3.6 Reconfigurer et supprimer un élément

L'instance récupérée lors de la création d'un élément d'un widget 'Canvas' permet de manipuler cet élément.

Suppression d'item

canevas0.delete(rectangle1) détruit l'élément **rectangle1**

Déplacements

canevas0.move(rectangle, 20, -10) déplace un élément de Canvas

canevas0.coords(image0, (50, 100)) modifie les coordonnées d'un élément de Canvas (voir **images**)

Hiérarchie

Chaque élément créé dans le widget 'Canvas' recouvre les précédents. Pour modifier la hiérarchie de la pile:

canevas0.tag_lower(ovale0) envoie l'élément 'ovale0' tout en bas de la pile (remplace l'ancien **lower**)

canevas0.tag_raise(rectangle0) envoie l'élément 'rectangle0' tout en haut de la pile (remplace l'ancien **lift**)

```
#!/usr/bin/python
import Tkinter

racine0=Tkinter.Tk()
racine0.geometry("300x200")

canevas0=Tkinter.Canvas(racine0, width=300, height=200)
canevas0.pack()
rectangle0=canevas0.create_rectangle(50, 50, 150, 100, fill="red")
ellipse0=canevas0.create_oval(75, 75, 150, 125, fill="blue") # se place au-dessus
canevas0.tag_lower(ellipse0) # replace le rectangle devant l'ellipse

racine0.mainloop()
```

itemconfigure

canevas0.itemconfigure(ellipse0, fill="green") colore l'objet **ellipse0** en vert. L'exemple suivant permet de changer la couleur d'un rectangle par un clic sur un bouton:

```
racine0=Tkinter.Tk()
def rouge():
    canevas0.itemconfig(rectangle0, fill='red')
def vert():
    canevas0.itemconfig(rectangle0, fill='green')
def bleu():
    canevas0.itemconfig(rectangle0, fill='blue')
canevas0=Tkinter.Canvas(racine0, width=300, height=200, background='darkgray')
canevas0.pack()
rectangle0=canevas0.create_rectangle(50, 50, 250, 150, fill='gray')
bouton1=Tkinter.Button(racine0, text="Rouge!", command=rouge)
bouton1.pack(side=Tkinter.LEFT)
bouton3=Tkinter.Button(racine0, text="Bleu!", command=bleu)
bouton3.pack(side=Tkinter.RIGHT)
bouton2=Tkinter.Button(racine0, text="Vert!", command=vert)
bouton2.pack()
racine0.mainloop()
```

- **itemconfig()** peut remplacer **itemconfigure()**
- puisqu'il y a de la place, il est possible de plaquer le bouton "Rouge" à gauche, le "Bleu" à droite et puis le "Vert" au milieu (par défaut). Malgré les **side=**, modifier l'ordre de définition des boutons change le résultat. **grid()** peut être utilisé pour faciliter le placement des widgets.

4. Widgets complexes

4.1 OptionMenu Nv 2014.11

Ce premier widget est une composition d'un bouton et d'un menu, qui permet de choisir une option dans une liste. C'est pourtant un des widgets les plus simples ne contenant pas beaucoup d'options.

```
import Tkinter

def ecran(dummy):
    print choix0.get()

racine0=Tkinter.Tk()

choix0=Tkinter.StringVar(); choix0.set("Rouge")
option0=Tkinter.OptionMenu(racine0, choix0, "Rouge", "Vert", "Bleu", command=ecran)
option0.pack()

racine0.mainloop()
```

- les choix sont données à la suite l'un de l'autre
- `choix0.set("Rouge")` détermine la valeur par défaut

4.2 PanedWindow

PanedWindow permet de diviser une fenêtre en plusieurs panneaux adaptables.

```
import Tkinter
racine0=Tkinter.Tk()
racine0.geometry("400x300")
division0=Tkinter.PanedWindow(orient=Tkinter.VERTICAL)
division0.pack(expand="yes", fill="both")
panneau1=Tkinter.Label(division0, text="Panneau Un")
division0.add(panneau1)
panneau2=Tkinter.Label(division0, text="Panneau Deux")
division0.add(panneau2)
panneau3=Tkinter.Label(division0, text="Panneau Trois")
division0.add(panneau3)
racine0.mainloop()
```

On adapte cette fonction dans l'autre direction avec les paramètres suivants: `orient=Tkinter.HORIZONTAL`.

Il est possible de créer des subdivisions dans un des panneaux. Dans l'exemple suivant, c'est le panneau `bas0` qui devient l'objet à diviser par PanedWindows: c'est donc à lui que les sous-panneaux `gauche` et `droite` doivent se référer.

```
import Tkinter
racine0=Tkinter.Tk()
racine0.geometry("400x300")
division0=Tkinter.PanedWindow(orient=Tkinter.VERTICAL)
division0.pack(expand="yes", fill="both")
haut0=Tkinter.Label(division0, text="Panneau du haut")
division0.add(haut0)
milieu0=Tkinter.Label(division0, text="Panneau du milieu")
division0.add(milieu0)
bas0=Tkinter.PanedWindow(orient=Tkinter.HORIZONTAL) # nouvelle division
bas0.pack(expand="yes", fill="both")
gauche=Tkinter.Label(bas0, text="Panneau bas-gauche")
bas0.add(gauche)
droit=Tkinter.Label(bas0, text="Panneau bas-droit")
bas0.add(droit)
division0.add(bas0) # on acheve la declaration du panneau bas
racine0.mainloop()
```

4.3 Listbox

Le script suivant permet le transfert dans la zone texte d'un mot dans une liste, par un double clic gauche.

```
import Tkinter

racine0=Tkinter.Tk()
liste0=Tkinter.Listbox(racine0, width=10)
liste0.pack()
texte0=Tkinter.Text(racine0, width=10)
texte0.pack()

for element in ["Monthy", "Python", "Flying", "Circus"]:
    liste0.insert(Tkinter.END, element)
def clic(inutile):
    texte0.insert(Tkinter.INSERT, liste0.get(liste0.curselection())+" ")

liste0.bind('<Double-1>', clic)
racine0.mainloop()
```

La boucle `for` remplit la liste des éléments dans la Listbox.

`inutile` est une variable nécessaire mais qu'on n'utilise pas.

`insert()` permet d'ajouter l'élément cliqué à l'endroit du curseur, défini par `Tkinter.INSERT`, `END` pour la fin du texte, `CURRENT` pour le début)

Il est possible de remplacer le double-clic par une confirmation par bouton:

```
import Tkinter
racine0=Tkinter.Tk()

liste0=Tkinter.Listbox(racine0, width=10, selectmode=Tkinter.MULTIPLE)
liste0.pack()

bouton0=Tkinter.Button(racine0, text='Confirmer')
bouton0.pack()

texte0=Tkinter.Text(racine0, width=10)
texte0.pack()

for element in ["Monthy", "Python", "Flying", "Circus"]:
    liste0.insert(Tkinter.END, element)

def clic(inutile):
    for i in liste0.curselection():
        texte0.insert(Tkinter.INSERT, liste0.get(i)+" ")

bouton0.bind('<Button-1>', clic)
racine0.mainloop()
```

De plus, `Listbox` accepte `selectmode=Tkinter.MULTIPLE` pour un mode de sélection multiple en cliquant successivement sur plusieurs

items, et **EXTENDED** qui permet Ctrl-Clic pour une succession d'items et Maj-Clic pour une suite d'items consécutifs. Il a fallu modifier la fonction **clic** pour qu'elle accepte une réponse multiple, sous forme de tuple contenant les index des items choisis.

Le mode par défaut est **selectmode=Tkinter.SINGLE**, qui fonctionne mal avec "<Button-1>" (simple clic gauche): l'élément cliqué arrive avec un coup de retard.

Pour être complet, il faudrait également parler de **selectmode=Tkinter.BROWSE** censé déplacer un item en le tirant (drag'n drop), mais cela ne semble pas fonctionner.

4.4 Scrollbar Rév. 2014.10

Scrollbar permet de faire défiler dans une surface limitée quelques widgets, comme **Text**, **Entry**, **Listbox** et, avec une méthode quelque peu **différente**, **Canvas**.

```
import Tkinter

fenetre0=Tkinter.Tk()

ascenseur0=Tkinter.Scrollbar(fenetre0)
ascenseur0.pack(side=Tkinter.RIGHT, fill=Tkinter.Y)

texte0=Tkinter.Text(fenetre0, yscrollcommand=ascenseur0.set)
texte0.insert(Tkinter.END, "bla bla bla "*443)
texte0.pack(side=Tkinter.LEFT, fill=Tkinter.BOTH)

ascenseur0.config(command=texte0.yview)

fenetre0.mainloop()
```

- **orient=Tkinter.VERTICAL** est l'orientation par défaut de **Scrollbar**
- **fill=Tkinter.Y** met de l'espace entre les deux flèches de l'ascenseur horizontal
- **side=Tkinter.LEFT** aurait placé l'ascenseur à gauche

Text et **Scrollbar** se réfèrent l'un à l'autre. Il convient de d'abord définir **Scrollbar** en récupérant sa variable de création (dans cet exemple, **ascenseur0**), de définir le texte en s'y référant avec **yscrollcommand= ascenseur0.set**, puis de reconfigurer l'ascenseur avec **command=texte0.yview**

Il était possible de garder le défilement vertical du texte avec un ascenseur horizontal, en changeant ses deux lignes de définitions:

```
import Tkinter

fenetre0=Tkinter.Tk()

# ces deux lignes changent:
ascenseur0= Tkinter.Scrollbar(fenetre0, orient=Tkinter.HORIZONTAL)
ascenseur0.pack(side=Tkinter.TOP, fill=Tkinter.X)

texte0=Tkinter.Text(fenetre0, yscrollcommand=ascenseur0.set)
texte0.insert(Tkinter.END, "bla bla bla "*443)
texte0.pack(side=Tkinter.LEFT, fill=Tkinter.BOTH)

ascenseur0.config(command=texte0.yview)

fenetre0.mainloop()
```

Par contre, la commande **texte0.yview** doit rester la même, puisque le défilement reste vertical

On utilise indifféremment les constantes **Tkinter.Y**, **Tkinter.N**, **Tkinter.VERTICAL** et **Tkinter.HORIZONTAL** et leurs valeurs-chaînes "y", "n", "vertical" ou "horizontal"

Ascenseur avec Canvas

Pour le widget **Canvas**, c'est un peu plus compliqué. Premièrement, il semble qu'il faille l'inclure dans un cadre **Frame()**. Deuxièmement, le paramètre **scrollregion=()** doit préciser les dimensions de la page entière concernée par le défilement. Merci à effbot.org/tkinterbook/ pour cette précision.

```
import Tkinter
fenetre0=Tkinter.Tk()

cadre0=Tkinter.Frame(fenetre0, width=300, height=300)
cadre0.pack()
canevas0=Tkinter.Canvas(cadre0, bg='#FFFFFF', width=250, height=200, scrollregion=(0, 0, 250, 250))

ascenseur0=Tkinter.Scrollbar(cadre0)
ascenseur0.pack(side=Tkinter.RIGHT, fill=Tkinter.Y)
ascenseur0.config(command=canevas0.yview)

canevas0.config(width=250, height=200)
canevas0.config(yscrollcommand=ascenseur0.set)
canevas0.pack(side=Tkinter.LEFT, expand=True)

texte=""
T'es fou
Tire pas
C'est pas des corbeaux
C'est mes souliers
Je dors parfois dans les arbres
"""
textel=canevas0.create_text(text=texte, 12, 12, width=280, anchor=Tkinter.NW)
texte2=canevas0.create_text(12, 200, text="'Moi dans l'arbre'\nPaul Vicensini", width=280, anchor=Tkinter.NW)

fenetre0.mainloop()
```

4.5 Menu

Voici un exemple commenté d'un système de menu fonctionnel comportant des cascades (sous-menus). Pour la fonction `Menu`, deux méthodes sont nécessaires: `add_cascade` pour ajouter un menu ou un sous-menu, et `add_command` pour décider de la commande associée au clic. Les actions sont ici limitées à l'affichage d'un texte.

```
import Tkinter
racine0=Tkinter.Tk()

texte0=Tkinter.Text(racine0) # prevoit une place pour l'affichage des textes
texte0.pack()

def ecran(var): # fonction servant a l'affichage des textes:
    texte0.insert(Tkinter.END, var)

sysdemenu0=Tkinter.Menu(racine0) # Creation du systeme de menu

menu1=Tkinter.Menu(sysdemenu0, tearoff="0") # Creation du premier menu:
sysdemenu0.add_cascade(label="Menu 1", menu=menu1)

# addition des deux items pour le premier menu et leur commande associee
menu1.add_command(label="Credit", command=lambda: ecran('Credit: www.jchr.be\n'))
menu1.add_command(label="Quitter", command=racine0.quit)

menu2=Tkinter.Menu(sysdemenu0) # Creation du second menu
sysdemenu0.add_cascade(label="Menu 2", menu=menu2)

# addition du premier item pour le second menu et leur sous-items associes
item1=Tkinter.Menu(menu2)
menu2.add_cascade(label="Item 1", menu=item1)

# addition des sous-items du premier item du second menu et leur commande associee
item1.add_command(label="Action 1", command=lambda: ecran('Item 1 / Action 1\n'))
item1.add_command(label="Action 2", command=lambda: ecran('Item 1 / Action 2\n'))

item2=Tkinter.Menu(menu2) # addition du second item pour le second menu et leur sous-items associes
menu2.add_cascade(label="Item 2", menu=item2)

# addition des sous-items du second item du second menu et leur commande associee
item2.add_command(label="Action 1", command=lambda: ecran('Item 2 / Action 1\n'))
item2.add_command(label="Action 2", command=lambda: ecran('Item 2 / Action 2\n'))
item2.add_command(label="Action 3", command=lambda: ecran('Item 2 / Action 3\n'))
racine0.config(menu=sysdemenu0)
racine0.mainloop()
```

Par défaut, chaque menu commence par une ligne discontinue, et un clic sur celle-ci transfère le menu dans une petite fenêtre indépendante. Pour supprimer cette ligne et cette possibilité, ajouter le paramètre `tearoff=0` dans la fonction `Menu()`, comme cela a été fait dans l'exemple pour `menu1`.

5. Modules

5.1 mainloop / quit

L'interface graphique Tkinter contient une boucle ouverte par `racine0=Tkinter.Tk()` et fermée par `racine0.mainloop()`. Le nom de la variable `racine0`, qui définit le widget principal (une fenêtre) est arbitraire, les anglophones semblent en général l'appeler `root`. Il est possible de quitter l'interface graphique en cliquant le [X] en haut et à droite, mais il est possible (et conseillé) d'associer la sortie à une commande qui demande par exemple de sauvegarder quelque chose. On utilise pour ce faire la commande `quit`, généralement associée à un bouton.

```
import Tkinter

racine0=Tkinter.Tk()

bouton0=Tkinter.Button(racine0, text="Quitter", command=racine0.quit)
bouton0.pack()

racine0.mainloop()
```

5.2 pack, grid et place Rév. 2014.11

À part `Tk()`, qui utilise `mainloop()`, et `TopLevel()`, qui en est exempté, les fonctions de création d'objet doivent être confirmées par une méthode spéciale, à choisir entre `pack()`, `grid()` et `place()`.

`pack()`

Les exemples en général très simples de cette page contiennent le plus souvent la méthode `pack()`, que l'on utilise pour l'affichage effectif du widget, précisé par sa variable de création: `variable0.pack()`.

- `side=` permet de diriger le widget vers une direction dans son parent, avec les constantes `Tkinter.RIGHT`, `Tkinter.LEFT`, `Tkinter.TOP` ou `Tkinter.BOTTOM`, ou les chaînes "right", "left", "top" ou "bottom".
- `fill=` permet à un widget d'utiliser l'espace laissé par le parent, respectivement en largeur avec `Tkinter.X`, en hauteur avec `Tkinter.Y`, dans les deux dimensions avec `Tkinter.BOTH` ou les chaînes "x", "y", "both". Par défaut, il s'agit de `Tkinter.None` ou "none".
- `expand=` permet (1) ou non (0, par défaut) à un widget d'utiliser le maximum d'espace
- `in_ =` désigne un autre widget que le parent comme widget de référence. Cet autre widget doit nécessairement être descendant du parent

`grid()`

`grid` permet de placer des widgets dans les cases d'une grille, selon les coordonnées rangée (`row`) / colonne (`column`). Ne jamais mélanger `pack()` et `grid` dans un même conteneur ("frame" ou fenêtre-racine).

```
import Tkinter
```

```
def afficher(): # interroge et affiche la valeur retour
    print retour0.get()

racine0=Tkinter.Tk()
cadre0=Tkinter.Frame(racine0)

# titre pour chaque ligne
Tkinter.Label(cadre0, text=" 1er").grid(row=0, column=0)
Tkinter.Label(cadre0, text=" 2nd").grid(row=0, column=1)
Tkinter.Label(cadre0, text=" 3e").grid(row=0, column=2)

retour0=Tkinter.IntVar()
bouton1=Tkinter.Radiobutton(cadre0, text="Oui", variable=retour0, value=1)
bouton2=Tkinter.Radiobutton(cadre0, text="Bof", variable=retour0, value=3)
bouton3=Tkinter.Radiobutton(cadre0, text="Non", variable=retour0, value=2)

bouton1.grid(row=1, column=0)
bouton2.grid(row=2, column=1)
bouton3.grid(row=1, column=2)

cadre0.pack(side=Tkinter.TOP)

bouton0=Tkinter.Button(racine0, text="Cliquer", command=afficher)
bouton0.pack(side=Tkinter.BOTTOM)

racine0.mainloop()
```

place()

place() est un mode assez simple de positionnement pour un nombre restreint de widgets.

Exemple à intégrer...

- **in_**= un widget est en général placé par rapport à son parent. **in_** permet de désigner un autre widget (un descendant du parent?)
- **bordermode=** Tkinter.OUTSIDE précise que la taille et la position du widget sont relatives à la taille extérieure du widget de référence ; par défaut, Tkinter.INSIDE: taille et position relatives à la surface intérieure
- **width=** et **height=** taille du widget en pixels ; par défaut, la taille prévue par Tkinter
- **relwidth=** et **relheight=** («réels» de 0.0 à 1.0) taille, relative au widget de référence
- **x=** et **y=**: position par rapport à la gauche et au haut du widget, en px
- **relx=** et **rely=** («réels» de 0.0 à 1.0) position, relative à la taille du widget de référence
- **anchor=** spécifie le côté ou le sommet du widget à placer à la position définie. Le défaut est Tkinter.NW (voir **alignements**)

Les positionnements étant individuels, l'usage de **place()** devient vite ingérable avec l'accroissement du nombre de widgets.

5.3 Modifier Rév. 2014.11

configure() ou **config()** permet de modifier complètement un paramètre (dans l'exemple, la chaîne affichée par un «Label») et/ou d'en ajouter (la **couleur** du texte et celle du fond).

```
import Tkinter

def colorer():
    texte0.config(text="Ce texte change et prend de la couleur", fg="blue", bg="red")

racine0=Tkinter.Tk()

texte0=Tkinter.Label(racine0, text="Ceci est un texte en noir et blanc")
texte0.pack()

bouton0=Tkinter.Button(racine0, text="Colorer le texte", command=colorer)
bouton0.pack()

racine0.mainloop()
```

Cela peut être utilisé pour basculer de l'état **DISABLED** d'un widget à son état **ACTIVE**.

```
import Tkinter

def activer():
    bouton0.config(state=Tkinter.ACTIVE)

racine0=Tkinter.Tk()

bouton0=Tkinter.Button(racine0, text="Quitter", state=Tkinter.DISABLED, command=racine0.quit)
bouton0.pack()

bouton1=Tkinter.Button(racine0, text="Activer", command=activer)
bouton1.pack()

racine0.mainloop()
```

Les widgets en état **DISABLED** ne sont pas tous grisés. Il faut parfois agir sur les paramètres de couleurs (**fg**, **bg**...) , ce qui peut se faire par le même **config()**.

5.4 détruire Rév. 2014.10

destroy() supprime un widget. Dans l'exemple ci-dessous, le bouton en appelle à son autodestruction (**racine0** doit être défini avant la fonction **détruire()**):

```
import Tkinter
racine0=Tkinter.Tk()
def detruire():
```



```
bouton0.destroy()
texte0=Tkinter.Label(racine0, text="Destruction accomplie")
texte0.pack()
bouton0=Tkinter.Button(racine0, text="Autodestruction", command=detruiure)
bouton0.pack()
racine0.mainloop()
```

5.5 événements Nv. 2014.11

bind lie un événement (clic de souris, touche...) à une action, l'événement à attendre est codé par une chaîne commençant par < et terminant par >.

Événements liés au clavier

L'exemple suivant attend un clic de touche et l'affiche dans un label:

```
import Tkinter

def touche(evt):
    texte0.config(text=evt.char)

racine0=Tkinter.Tk()

texte0=Tkinter.Label(racine0, text="...", font="Arial 64", width=3)
texte0.pack()
racine0.bind("<Key>", touche)

racine0.mainloop()
```

- pour réceptionner toutes les touches: **<Key>** ou **<KeyPress>**
- la fonction **toucher()** (appelée sans parenthèses) récupère l'événement dans une variable (ici: **evt0**) et l'interroge par une méthode (ici: **evt0.char**)
- **<KeyPress-w>** ou **<w>** attend le caractère **w** (minuscule). Tous les caractères sont permis, mais **<space>** convient pour attendre la barre-espace, **<less>** pour <, **<Return>** (mais **<Enter>** est réservé à la souris: voir exemple suivant) pour la touche Enter
- Les modificateurs **Control**, **Alt**... peuvent être combinés: **<Control-Alt-w>** attend cette combinaison de touche. **<Any-w>** permet n'importe combinaison de touche modificatrices, mais **shift** ne semble pas fonctionner (en tout cas sous Linux: utiliser une majuscule).
- **evt0.keycode** renvoie le numéro de touche, **evt0.keysym** un caractère ou sa description, ou une description de la touche, comme (liste non exhaustive):
 - Alt_L, BackSpace, Cancel ([Break]), Caps_Lock, Control_L, Control_R, Delete, Down (flèche), End, Escape, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Home, Insert, ISO_Level3_Shift ([AltGr]), Left (flèche), Menu, Next ([Pg Dn]), Num_Lock, Pause, Print, Prior, Return ([Enter]), Right (flèche), Scroll_Lock, Shift_L, Super_L ([Win]), Tab ([Pg Up]), Up (flèche)
 - agrave, ampersand, apostrophe, asterisk, at [@], backslash, backslash, bar, braceleft [{], braceright [}], bracketleft [()], bracketright [()], ccedilla, colon [:], comma, dead_acute ['], dead_diaeresis ['], dead_grave ['], dead_tilde [~], degree, dollar, eacute, egrave, equal, exclam, guillemotleft [«], guillemotright [»], minus, parenleft ([(]), parenright ([)]), percent, period, plus, question, quotedbl ["], section [§], semicolon [;], slash, sterling [£], threesuperior, twosuperior, underscore
 - Avec NumLock: KP_0, KP_1, KP_2, KP_3, KP_4, KP_5, KP_6, KP_7, KP_8, KP_9, KP_Decimal (point)
 - Sans NumLock: KP_Add ([+]), KP_Begin ([Home]), KP_Delete, KP_Down, KP_End, KP_Home, KP_Insert, KP_Left, KP_Next ([PgUp]), KP_Prior ([PgDn]), KP_Right, KP_Up
 - Avec ou sansNumLock: KP_Divide, KP_Enter, KP_Multiply, KP_Subtract, KP_Subtract
- **evt0.type** renvoie le type d'événement: 2 pour un appui sur une touche (également en mode «répétition»), 3 pour une touche que l'on quitte (également en mode «répétition»)

Événements liés à la souris

Les clics et la position de la souris peuvent être déterminés lors d'événements sur un widget. En guise d'exemple, pour récupérer les coordonnées de la souris lors d'un clic:

```
import Tkinter

def souris(evt):
    print "X: %03d - Y: %03d" %(evt.x_root, evt.y_root)
    print "x: %03d - y: %03d" %(evt.x, evt.y)
    print evt.num, evt.type, evt.widget

racine0=Tkinter.Tk()

cadre0=Tkinter.Frame(racine0, width=400, height=300)
cadre0.bind("<Button-1>", souris)
cadre0.pack()

racine0.mainloop()
```

- **bind()** lie, pour un widget (défini par sa variable de création), une action à une fonction
- **<Button-1>** est l'action de cliquer sur le bouton gauche de la souris(voir liste plus bas)
- la fonction **souris** récupère l'événement (ici, par la variable **evt0**) et en exprime les attributs:
 - **.x** et **.y** renvoient les coordonnées de la souris par rapport au widget survolé
 - **.x_root** et **.y_root** renvoient les coordonnées de la souris par rapport à l'écran
 - **.num** renvoie le numéro du bouton cliqué
 - **.type** renvoie le type d'événement: 4 pour un (ou double, triple) clic, 5 pour un retour de clic, 6 pour un mouvement, 7 pour une entrée sur un widget, 8 pour une fin de survol
 - **.widget** retourne l'identifiant du widget concerné, sous la forme .140524171117416
- **.width** et **.height** renvoient les nouvelles dimensions d'un widget (en cas de reconfiguration)

Les actions de la souris peuvent être:

- **<Button>** pour n'importe quel bouton, **<Button-1>** ou **<1>** pour le bouton de gauche et **<Button-3>** ou **<3>** pour le bouton de droite
- **<Button-2>** ou **<2>** pour le bouton du milieu (s'il existe, ou les boutons gauche et droit en même temps)
- **<ButtonRelease>**, **<ButtonRelease-1>**, **<ButtonRelease-2>**... le déclenchement a lieu avec le retrait du clic

- **<Motion>** permet un déclenchement lorsque la souris bouge en survolant un widget ; **<B1-Motion>** attend un clic et un mouvement
- **<Double-Button-1>**, **<Triple-Button-1>**: attend deux ou trois clics rapprochés (à ne pas utiliser avec une autre procédure pour un simple clic sur le même widget, qui se déclencherait également).
- **<Enter>** déclenche une procédure lorsque la souris survole un widget, **<Leave>** lorsque la souris quitte le widget

5.6 after (temporalisation) Nv. 2014.11

La méthode **after()** permet de réaliser des tâches en fonction du temps sans bloquer le script, comme le ferait la commande **time.sleep()**.

```
import Tkinter, time

def temps():
    global tps0
    heure=time.strftime("%Y.%m.%d - %H:%M:%S")
    texte0.config(text=heure)
    tps0=racine0.after(1000, temps)

def stop():
    global tps0
    racine0.after_cancel(tps0)

racine0=Tkinter.Tk()

texte0=Tkinter.Label(racine0, text="")
texte0.pack()

bouton0=Tkinter.Button(racine0, text="Stop", command=stop)
bouton0.pack()
temps()

racine0.mainloop()
```

Une fois la fonction **temps()** lancée, elle s'appelle avec régularité, sans empêcher les autres widgets d'être appelables.

- le nombre précisé en premier paramètre précise le nombre de millisecondes entre chaque appel
- La variable **tps0** permet de terminer la temporalisation avec **after_cancel()**.

Un autre exemple **ici**

À suivre...

bell ne semble pas fonctionner ; peut-être est-ce dû à la configuration du système ou de la console.

6. Styles

En important Tkinter avec **import Tkinter** (comme dans les exemples de cette page), les constantes du système doivent être préfixée de **Tkinter**, par exemple **Tkinter.HORIZONTAL**. Avec un alias, par exemple **import Tkinter as TK**, la valeur s'appelle avec le préfixe **TK**: **TK.HORIZONTAL**. Avec le mode d'importation **from Tkinter import *** le préfixage est supprimé: la constante est tout simplement **HORIZONTAL**.

Quel que soit le mode d'importation (et donc de préfixage), la valeur d'une constante est généralement remplaçable par la même chaîne sans préfixage, en minuscules et avec guillemets, soit, pour notre exemple, **"horizontal"**:

importation du module	nom de constante	valeur
import Tkinter	Tkinter.HORIZONTAL	"horizontal"
import Tkinter as alias	alias.HORIZONTAL	
from Tkinter import *	HORIZONTAL	

Il existe quelques exceptions: **SEL_FIRST** vaut **"sel.first"** et **SEL_LAST** vaut **"sel.last"**

Quelques constantes contiennent une valeur numérique, qui s'écrit donc sans guillemets: **FALSE**, **NO** et **OFF** valent 0, et **TRUE**, **YES** et **ON** valent 1

READABLE vaut 2 - **WRITABLE** vaut 4 - **EXCEPTION** vaut 8

6.1 Alignements

side= pour **pack()** et les **ascenseurs**, ou **justify=""** pour les textes, acceptent **Tkinter.TOP**, **Tkinter.RIGHT**, **Tkinter.BOTTOM** et **Tkinter.LEFT**, ou le contenu de ces constantes **"top"**, **"right"**, **"bottom"** et **"left"**.

compound (voir **Boutons**) accepte en outre les constantes **Tkinter.CENTER** et **Tkinter.NONE**, ou leur contenu **"center"** et **"none"**.

anchor=, utilisé dans le positionnement des **images** reçoit les constantes basées sur les points cardinaux: **Tkinter.N**, **Tkinter.NE**, **Tkinter.E**, **Tkinter.SE**, **Tkinter.S**, **Tkinter.SW**, **Tkinter.W**, **Tkinter.NW**, **Tkinter.CENTER** ou leur contenu: **"n"**, **"ne"**, **"e"**, **"se"**, **"s"**, **"sw"**, **"w"**, **"nw"** et **"center"**.

orient= reçoit les constantes **Tkinter.VERTICAL** ou **Tkinter.HORIZONTAL**, qui correspondent aux chaînes **"vertical"** ou **"horizontal"**.

6.2 Reliefs

La majorité des widgets peuvent afficher un type de relief avec le paramètre **relief=** défini par les constantes **Tkinter.RAISED** (élevé), **Tkinter.SUNKEN** (enfoncé), **Tkinter.FLAT** (plat, par défaut), **Tkinter.GROOVE** (rainure) ou **Tkinter.RIDGE** (crête), ou leur contenu: **"raised"**, **"sunken"**, **"flat"**, **"groove"** ou **"ridge"**. Les simples **boutons** à cliquer disposent déjà du type **"raised"** avec une animation **"sunken"** lors du clic gauche.



borderwidth= ou **bd=** permet de préciser la grosseur des traits. L'image ci-contre a été réalisée avec **bd=3** (ou **bd="3"**).

6.3 Fontes Rév. 2014.11

Valable pour **Label**, **Text** et **create_text**, il y a plusieurs manières d'imposer une fonte, une hauteur et une décoration:

font=("Courier", "16", "bold italic")

```
font="Courier 30 bold"
font=("-*-Helvetica-medium-r-*-*--200-*-*-*-*")
```

```
import Tkinter
racine0=Tkinter.Tk()
etiquette0=Tkinter.Label(racine0, text="Texte gras et italique", font="Courier 32 bold underline")
etiquette0.pack()
racine0.mainloop()
```

Le paramètre **font** permet de préciser (dans une seule chaîne, sans virgule)

- une police de caractère
 - un type: **serif** (avec empattement), **sans** (sans empattement), **monospace** (largeur fixe)
 - une fonte installée sur votre système: **times**, **helvetica** ou **arial**, **courier**, ...
 - une fonte utilisée par Tkinter - très petites: TkTooltipFont et TkSmallCaptionFont - moyennes: TkDefaultFont, TkTextFont, TkIconFont, TkMenuFont, TkFixedFont (monospace), TkHeadingFont (gras) - plus grande: TkCaptionFont
- une hauteur de caractère en points d'imprimerie, 72e de pouce (25, 4mm), mais semble en fait être la hauteur en pixels: sur un écran de 100px/pouce, cela apparaîtra plus petit
- une graisse: **bold**, ou **normal** (par défaut)
- un style: **italic**, ou **roman** (normal, par défaut)
- une décoration: **underline**, **overstrike**

```
TkTooltipFont
TkSmallCaptionFont
TkDefaultFont
TkTextFont
TkIconFont
TkMenuFont
TkHeadingFont
TkCaptionFont
TkFixedFont
```

6.4 Couleurs

On utilise la forme "#RRGGBB" où les lettres représentent le rouge, le vert et le bleu en chiffres hexadécimaux (de 0 à 9 puis de a à f). Cette chaîne est à fournir aux paramètres **foreground=""**, **background=""**, **fill=""** et **outline=""**

Il existe des couleurs toutes faites parmi lesquelles:

```
white, black, gray1 à gray99, light-gray, dimgray, gray, dark-gray
red (~2 / ~3 / ~4), pink, deep pink, magenta, violet, purple
yellow, gold, orange
green (dark~ / lawn ~ / lime ~ / forest ~ / yellow ~), olive drab
blue (medium ~ / midnight ~ / steel ~ / dark~ / deep~ / light~ / sky~), navy, blue, cyan
```

Voir également la boîte de **choix de couleur**.

6.5 Curseurs graphiques Ajout 2014.10

Différents curseurs graphiques sont possibles lorsqu'on survole un widget. La règle générale est de spécifier **cursor=""** lors de la définition du widget. Sans rien préciser, un widget adopte le curseur de son parent.

Certains widgets imposent leur curseur, comme **xterm** lorsqu'on survole un champs éditable, d'autres sont imposés par le système, comme le bureau Gnome a remplacé la montre bracelet de "watch" par une rosace tournante. Ils ne sont pas tous très réussis, ceux écrits en gras sont les plus intéressants ou élégants.

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**, **crosshair**, cross_reverse, diamond_cross, **dot**, dotbox, double_arrow, draft_large, draft_small, draped_box, exchange, **fleur** (4 directions), gobbler, gumby, **hand1**, **hand2**, **heart**, icon, iron_cross, leftbutton, left_ptr, left_side, left_tee, ll_angle, lr_angle, man, middlebutton, mouse, **pencil**, **pirate**, **plus**, **question_arrow**, rightbutton, **right_ptr**, right_side, right_tee, 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, **sprayan**, star, target, **tcross**, top_left_arrow, top_left_corner, top_right_corner, top_side, top_tee, trek, ul_angle, umbrella, ur_angle, **watch**, **X_cursor**, **xterm**

7. Modules associés

Les quatre bibliothèques qui suivent permettent d'ouvrir des boîtes toutes faites, qui permettent d'avertir, de demander une confirmation, de préciser une valeur, de sélectionner un fichier ou même de choisir une couleur.

7.1 Boîtes à message Rév. 2014.11

Si le paquet Debian **python-tk** est installé: **import tkMessageBox** importe les fonctions de boîtes à messages. Le bouton par défaut (que l'on peut confirmer par [Enter]) est celui de gauche: [OK], [Yes] ou [Retry].

- title=""** donne un titre à la boîte de message ou de choix
- message=""** définit le message en gras, à l'intérieur de la boîte
- detail=""** permet un message secondaire, dans une fonte de plus petite taille
- default="no"** ou **default="cancel"** redéfinit le bouton par défaut
- icon** et **type**: voir **Reconfiguration des boîtes**

Retournent **ok**, bouton nécessairement par défaut:

```
import tkMessageBox
tkMessageBox.showinfo(title="Information", message="There's spam")
tkMessageBox.showwarning(title="Avertissement", message="There's no spam")
tkMessageBox.showerror(title="Erreur", message="There's no spam")
```

Retourne **yes** (par défaut) ou **no**; [tab] permet de changer de bouton sensible à [Enter]:

```
import tkMessageBox
q=tkMessageBox.askquestion(title="Question", message="Is there spam?")
print q
```

Retournent **True** ou **False** - ou encore **None** pour **askyesnocancel()**:

```
import tkMessageBox
oc=tkMessageBox.askokcancel()      # default="ok" - sinon, ajouter default="cancel"
print oc
yn=tkMessageBox.askyesno()         # default="yes" - sinon, ajouter default="no"
print yn
ync=tkMessageBox.askyesnocancel()  # default="yes" - sinon, ajouter default="no" ou "cancel"
print ync
```

```
rc=tkMessageBox.askretrycancel() # default="retry" - sinon, ajouter default="cancel"
print rc
```

Reconfiguration des boîtes

Changement d'icone:

```
import tkMessageBox
tkMessageBox.askyesno(title="Dubitatif", message="Il n'y a pas de spam", icon="info")
```

- **icon=""** permet de forcer le type d'icone **"error"**, **"info"**, **"question"** ou **"warning"**, quelle que soit le type de réponse possible. L'exemple ci-dessus permet un choix Oui/Non mais avec l'icone du **i** dans un phylactère.

Changement de boutons de choix:

```
import tkMessageBox
tkMessageBox.showerror(type="yesno")
```

- **type=** permet de forcer d'autres choix dans une boîte particulière avec les valeurs **"ok"**, **"yesno"**, **"okcancel"**, **"retrycancel"**, or **"yesnocancel"**. L'exemple suivant permet d'afficher la boîte d'erreur (la **x** blanche sur le disque rouge de **showerror**), mais avec un choix entre [Oui] et [Non]
- **"abortretryignore"**, inédit parmi les boîtes toutes faites, permet de renvoyer les choix **"abort"**, **"retry"** ou **"ignore"**

En python3, le paquet Debian à installer est **python3-tk** et **messagebox** est un sous-module de **tkinter**. Par exemple,

```
import tkinter.messagebox
q0=tkinter.messagebox.askquestion(title="Titre", message="Message")
print(q0)
```

7.2 Boîtes de saisie Rév. 2014.11

Si le paquet Debian **python-tk** est chargé, **import tkSimpleDialog** charge un module permettant l'affichage de boîtes de saisie de données. Bien que ces boîtes soient indépendante de **Tk()**, il faut ouvrir une fenêtre-racine.

```
import Tkinter, tkSimpleDialog

racine0=Tkinter.Tk()
texte0="???"
etiquette0=Tkinter.Label(text=texte0)
etiquette0.pack()
texte0=tkSimpleDialog.askstring("Espace de parole", "Exprimez-vous!")
etiquette0.config(text=texte0)
racine0.mainloop()
```

tkSimpleDialog.askinteger("Titre", "Invitation") permet de saisir un entier (uniquement les chiffres de 0 à 9, + et -)

```
import Tkinter, tkSimpleDialog

racine0=Tkinter.Tk()
etiquette0=Tkinter.Label(text="A lancer dans une console; fermer\ncette fenetre en fin de saisie")
etiquette0.pack()
texte0=tkSimpleDialog.askinteger("Un nombre entier", "un signe (+ ou -) et des chiffres")
racine0.mainloop()
print texte0+1000000000
```

tkSimpleDialog.askfloat("Titre", "Invitation") permet de saisir un «réel» (chiffres, point décimal . (pas de virgule), +, -, **E** et **e** pour la notation exponentielle, mais pas les **j** / **J** pour les nombres complexes (faut-il prévoir deux boîtes?))

On peut ajouter quelques options valables pour entiers, décimaux ou chaînes, où "ac" est inférieur à "d":

- **initialvalue=** détermine une valeur par défaut
- **minvalue=** et **maxvalue=** déterminent les valeurs minimale et maximale acceptables

En python3, le paquet Debian s'appelle **python3-tk** et **simplifiedialog** est un sous-module de **tkinter**. Par exemple:

```
import tkinter.simplifiedialog
racine0=tkinter.Tk()
chaine=tkinter.simplifiedialog.askstring("Exprimez-vous!", "Veuillez dire ce que vous pensez", parent=racine0)
racine0.mainloop()
```

7.3 Sélectionneur de fichier Rév. 2014.11

Si le paquet Debian **python-tk** est chargé: **import tkFileDialog**

```
import tkFileDialog
repertoire0=tkFileDialog.askdirectory() # pour choisir un repertoire
print repertoire0
fichier1=tkFileDialog.askopenfilename() # pour selectionner le nom d'un fichier a ouvrir
print fichier1
fichier2=tkFileDialog.asksaveasfilename() # pour selectionner le nom d'un fichier a sauvegarder
print fichier2
```

- **title=** donne un titre à la boîte
- le bouton [OK] de la boîte de sélection de répertoire renvoie le répertoire par défaut (celui du script), [Ouvrir] et [Sauvegarder] ne fonctionnent que si un répertoire, un fichier à ouvrir ou à sauvegarder ont été sélectionnés
- il n'y a pas de bouton par défaut et pas de **default=** pour le déterminer, mais [Enter] renvoie le répertoire par défaut pour la boîte de sélection de répertoire.
- le bouton [Annuler] ou la fermeture de la boîte par [x] en haut et à droite renvoient une chaîne vide "" si aucun répertoire ou fichier n'est sélectionné, le tuple vide () si un répertoire ou un fichier est sélectionné
- les variables **fichier1** et **fichier2** contiennent le nom du fichier à ouvrir ou à sauvegarder, sans traiter le chargement ou la sauvegarde du fichier, même si une boîte de dialogue s'ouvre indiquant qu'ils sont sur le point d'être effacés - pour le traitement même, voir **ici**.

- `multiple="true"` ou `multiple=True` - pour `askopenfilename()` seulement - permet le choix de plusieurs fichiers (avec [shft-clc] et [ctrl-clc], retournés sous forme de tuple contenant les noms de fichiers sélectionnés.

`filetypes=[]` permet de définir des associations permettant de filtrer les fichiers selon leur extension. Les tuples ("`chaîne`", "`.ext`") sont collectés dans une liste:

```
import tkinterFileDialog
association0=[("Fichiers 'texte'", ".txt"), ("Fichiers 'html'", ".htm"), ("Tous les fichiers", ".*")]
fichier1=tkFileDialog.askopenfilename(filetypes=association0)
print fichier1
```

Deux autres boîtes permettent de combiner choix de fichier et leur traitement, sauvegarde ou chargement:

`askopenfile(mode='r')` permet de combiner la navigation dans une arborescence, la sélection du nom et le chargement effectif d'un fichier (`id0` est ici l'identifiant du fichier, pas son nom ni son contenu):

```
import tkinterFileDialog
id0=tkFileDialog.askopenfile(mode='r', title="Chargement d'un fichier texte")
texte=id0.read()
id0.close()
print texte
```

`asksaveasfile(mode='w')` permet de combiner la navigation dans une arborescence, la sélection du nom et la sauvegarde effective d'un fichier:

```
import tkinterFileDialog
idl=tkFileDialog.asksaveasfile(mode='w', title="Sauvegarde d'un fichier texte")
idl.write("Un texte court")
idl.close()
```

- `id0` est ici un identifiant de fichier, pas son nom
- une boîte de dialogue avertit qu'un fichier du nom sélectionné existe et va être écrasé

filedialog est un sous-module de **tkinter**:

```
import tkinter.filedialog
fichier1=tkinter.filedialog.askopenfilename()
fichier2=tkinter.filedialog.asksaveasfilename(defaultextension="png")
repertoire=tkinter.filedialog.askdirectory()
```

7.4 Sélectionneur de couleur Rév. 2014.11

Si le paquet Debian `python-tk` est installé sur votre système, il est possible d'importer le module `import tkColorChooser()`, qui ne comporte qu'un widget: `askcolor()`

```
import tkColorChooser

(rouge0, vert0, bleu0), couleur0=tkColorChooser.askcolor("#123456", title="Couleur?")
print rouge0, vert0, bleu0, couleur0
```

- `"#123456"` est une **couleur** proposée à l'utilisateur. Il est possible de fournir une chaîne de couleur reconnue: "navy", "green", "white"... par défaut, c'est le gris léger "#d3d7cf" utilisé par défaut par Tkinter.
- `title=` (facultatif) donne un titre à la fenêtre de sélection des couleurs
- `parent=cadre0` (facultatif) devrait permettre de désigner un widget réceptionnant la fenêtre, mais ne semble pas fonctionner.

L'expression `(rouge0, vert0, bleu0), couleur0` est un tuple composé de deux éléments. Le premier est lui-même un tuple composé de trois valeurs: les composantes fondamentales (de 0 à 255): rouge signal, vert émeraude et bleu outremer.

La chaîne `#RRVVBB` retournée par la variable-chaîne hexadécimale `couleur0` peut retourner les valeurs des trois couleurs de base de cette manière:

```
r0, v0, b0 = int(couleur0[1:3], 16), int(couleur0[3:5], 16), int(couleur0[5:], 16)
```

où une chaîne hexadécimale (par exemple `"#8F3BA6"`) est découpée en trois chaînes de deux caractères (ici "8F", "3B" et "A6") converties en nombre décimal par `int("8F", 16)`, etc.

Il est possible de recueillir la couleur en un seul (tuple):

```
import tkColorChooser
tuple0=tkColorChooser.askcolor(initialcolor=(150, 200, 250), title="Choisir une couleur...")
print tuple0[0][0] # est la quantite de rouge
print tuple0[0][1] # est la quantite de vert
print tuple0[0][2] # est la quantite de bleu
print tuple0[1] # est la couleur sous la forme #RRVVBB (chaîne)
```

- `tkColorChooser.askcolor` admet le paramètre `initialcolor=(rouge1, vert1, bleu1)`, où rouge1, vert1 et bleu1 sont trois nombres de 0 à 255 qui définissent la couleur de base affichée par le nuancier. `initialcolor="#ce1431"` fonctionne également.
- `title=""` définit le titre de la boîte de sélection de couleur.

Attention! en cas de clic sur le bouton [Annuler], la fonction renvoie le tuple `(None, None)`, ce qui peut perturber le script qui s'attend à recevoir un tuple de type `((r, v, b), chaîne)`. Il faut donc se prémunir de ce problème avec un script de ce genre:

```
import tkColorChooser
rouge0, vert0, bleu0 =128, 120, 230 # couleur initiale
couleurs, chaîne=tkColorChooser.askcolor(initialcolor=(rouge0, vert0, bleu0))
if couleurs==None: # bouton [Annuler]
    rouge, vert, bleu=rouge0, vert0, bleu0
else:
    # bouton [OK]
    rouge, vert, bleu=couleurs[0], couleurs[1], couleurs[2]
print chaîne, rouge, vert, bleu # couleurs, avec ou sans modification
```

Note: il est possible de saisir la valeur pour chaque couleur fondamentale ou directement éditer la chaîne hexadécimale, mais il faut

impérativement saisir la ou les valeurs modifiées avec [Enter] ([Return] ne fonctionne pas).

En python3, le module `tkinter` contient le sous-module `colorchooser`, dont l'unique fonction s'appelle également `askcolor()`, qui s'utilise de la même manière:

```
import tkinter.colorchooser
(rouge0, vert0, bleu0), couleur0=tkinter.colorchooser.askcolor()
print(rouge0, vert0, bleu0, couleur0)
```

L'exemple ci-dessus montre qu'en python3, les valeurs des couleurs séparées `rouge0`, `vert0` et `bleu0` ne sont pas entières. Pour chacune, il faut les traiter de cette façon (en python3, `round(3.14)` ou `round(3.14, 0)` arrondit en véritable entier):

```
import tkinter.colorchooser
(rouge0, vert0, bleu0), couleur0=tkinter.colorchooser.askcolor()
print(round(rouge0), round(vert0), round(bleu0), couleur0)
```

A. Documentation

A.1 Documentation dans le logiciel python

Dans le mode interactif, obtenu en saisissant `python` dans une console et après avoir importé le module:

```
>>> import Tkinter

>>> print dir() liste les modules chargés
>>> print Tkinter.__file__ renvoie la localisation de Tkinter sur le système
>>> print dir(Tkinter) liste les instructions du module Tkinter
>>> print Tkinter.fct.__doc__ documente l'instruction précisée de Tkinter
```

La commande `help` informe sur une fonction d'un module importé:

```
>>> help(Tkinter.Button)
```

Pour générer un fichier lisible avec un éditeur de texte, saisir dans une console:

```
python -c "import Tkinter; help(Tkinter)" > Tk.txt
```

`python -c` interprète une chaîne, constituée ici de l'importation du module et du lancement de l'aide sur ce module. Au lieu d'utiliser la sortie standard (la console), `> Tk.txt` sauvegarde le texte dans le fichier `Tk.txt`

En python3, le module est rebaptisé `tkinter`. Il faudra alors saisir:

```
python3 -c "import tkinter; help(tkinter)" > tk.txt
```

A.2 Documentation sur votre système GNU/Linux

Si python est installé, saisir `man python` dans une console renseigne sur les différentes manières de lancer python.

Le script `/usr/bin/pydoc` permet la consultation d'information de fonctions, modules, mots-clés:

`pydoc -g` lance une interface graphique pour une navigation dans le système d'aide

A.3 Documentation sur Internet

- [Penser en Tkinter](#) (.fr) de Stephen Ferg
- [Construire une interface graphique pas à pas en Python](#) (.fr) doc orientée objet
- docs.python.org/lib/module-Tkinter.html (.en) doc officielle
- www.pythonware.com/library/ (.en)
- effbot.org/tkinterbook/ (.en) très complet tout en restant accessible
- infohost.nmt.edu/tcc/help/pubs/tkinter/web/ (.en) et [pdf](#)

A.4 Documentation sur papier

- John E. Grayson, [Python and Tkinter Programming](#), Manning Publications
- Alex Martelli & David Ascher, eds.: "python cookbook", o'reilly, 2002 (anglais) comporte peu de choses sur le module Tkinter.

haut