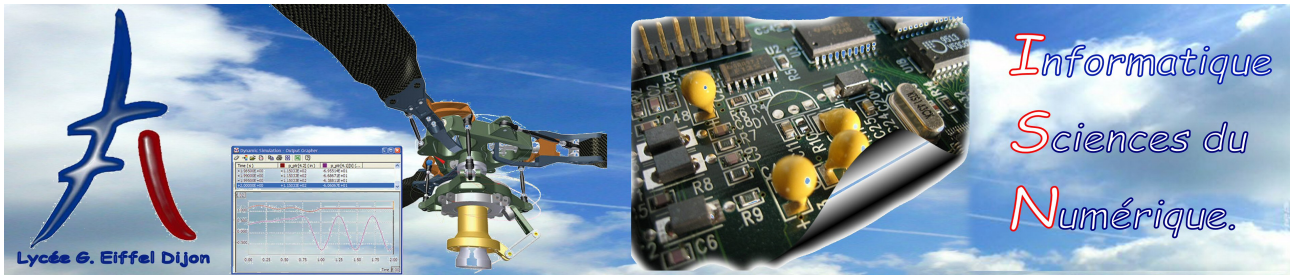

Documentation pour s'initier à Tkinter

Objectifs :

➢ **Débuter avec Tkinter**

1 Qu'est-ce que Tkinter ?

Il existe plusieurs modules pour réaliser des interfaces graphiques en Python comme Tkinter, PyQt ou PyGTK...
Nous nous intéresserons au module Tkinter qui est le plus utilisé et qui permet de créer facilement une fenêtre (windows, mac OS ou linux...) comportant par exemple une zone pour dessiner des images fixes ou animées, des boutons auxquels on relie des actions, une barre de menu, des cases à cocher...
Dans la suite, on utilisera le terme *méthode* à la place de *fonction* pour désigner une fonction car il est plus approprié pour le module Tkinter qui utilise la programmation orientée objet.
Nous utiliserons la syntaxe du module Tkinter de Python3, **qui peut différer de celle de Python2**.
Lorsqu'on écrit un module utilisant Tkinter, on doit importer celui-ci : par exemple en écrivant : *from tkinter import **.
De manière générale, on commencera par créer une fenêtre avec la syntaxe : *fenetre = Tk()*.
Tk() est une méthode (appelée constructeur) qui crée une fenêtre et l'affecte à la variable *fenetre* (on pourra choisir un autre nom) et grâce à cette variable, on pourra ensuite ajouter une zone de dessin, des boutons...
Le programme se terminera toujours par *fenetre.mainloop()*. Cette méthode lance une boucle qui permet d'écouter les événements clavier ou souris, etc... et envoie ceux-ci au programme principal, jusqu'à ce qu'un des événements mette fin à la boucle elle-même (comme par exemple un clic sur le bouton de fermeture de la fenêtre).
Dans le programme, on ajoutera à la fenêtre des *widgets* (window gadgets) comme des textes, des zones de saisie, des boutons, des menus...
Exemple le plus simple d'utilisation de Tkinter :

```
from tkinter import *
#On crée la fenêtre
fenetre = Tk()
#On crée un label (texte à afficher) qui a pour contenant la
    fenetre
texte = Label(fenetre, text="Hello world")
#On détermine la position géométrique du texte dans le contenant
#Ici, le contenant aura une taille adaptée au texte (méthode pack)
texte.pack()
fenetre.mainloop()
```

On importe le module *Tkinter*. On construit la fenêtre, on crée un *Label* (texte) en précisant que son conteneur est *fenetre* et en définissant le texte associé. On définit ensuite la position géométrique du *label* dans son conteneur : ici, avec la méthode *pack*, le conteneur (la fenêtre) adapte sa taille au contenu (le *label*). Finalement, on lance la boucle qui va permettre d'écouter les événements clavier ou souris.

2 Quelques Widgets

2.1 Le widget Label

Il permet d'afficher des textes (libellés). On peut choisir la couleur d'écriture (paramètre *fg*), changer la police de caractères (en créant une nouvelle police avec le module *font*), ajouter de l'espace entre le texte et le bord de la boîte contenant le texte (*padx*, *pady*)...

```
from tkinter import *
from tkinter import font

fenetre = Tk()
#On crée une nouvelle police de caractères
police = font.Font(family='Helvetica', size=36, weight='bold')
texte = Label(fenetre, text="Hello world", fg='red', font=police,
    relief=RIDGE)
#On met des marges autour de la boîte contenant le texte
texte.pack(padx=20, pady=30,)
fenetre.mainloop()
```

Plus d'explications :
http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/label.html

2.2 Le widget Entry

C'est une zone qui permet de saisir un texte ou un nombre. Il faut créer une variable qui va recevoir la valeur saisie (attention, les méthodes utilisées diffèrent selon que c'est une chaîne de caractères, un entier, un réel).

```
from tkinter import *


fenetre=Tk()
#Label pour expliquer ce qu'il faut saisir
description = Label(fenetre, text="Entrer une chaîne de caractères:
    ", width=40, height=3)
description.pack()
# Définition d'une variable chaîne pour recevoir la saisie d'un
    texte
resultat_saisie=StringVar()
#Création de la zone de texte
zone_texte = Entry(textvariable=resultat_saisie, width=40)
zone_texte.pack(padx=10, pady=20)
fenetre.mainloop()
# Affiche le texte saisi dans la console à la fermeture de la
    fenêtre
print(resultat_saisie.get())
```

Si la saisie est un entier, il faudra plutôt écrire : *resultat_saisie=IntVar()*.
Si c'est un réel, il faudra plutôt écrire : *resultat_saisie=DoubleVar()*.
Plus d'explications :
http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/entry.html

2.3 Le widget Button

C'est un bouton sur lequel on peut cliquer. Il est en général relié à une action à effectuer lors du clic, c'est-à-dire à une fonction.

```
from tkinter import *
fenetre = Tk()
#La fonction appelée en cas de clic est fenetre.destroy
bouton = Button(fenetre, text="Quitter", command=fenetre.destroy)
#On met des marges autour du bouton
bouton.pack(padx=20, pady=20)
fenetre.mainloop()
```

On peut définir sa propre fonction à exécuter en cas de clic sur le bouton. Attention, on doit passer au paramètre *command* le nom de la fonction sans guillemets ni parenthèses (ce n'est pas un appel de la fonction). Il est impossible de passer des arguments à la fonction appelée (celle-ci ne pourra d'ailleurs pas avoir de paramètres sauf si ils ont une valeur par défaut).

```
from tkinter import *

#On définit sa propre fonction à exécuter
#sans paramètre ou avec des paramètre qui ont tous une valeur par
    défaut
def fonction_clic() :
    print("Vous avez cliqué sur le bouton")

fenetre = Tk()
bouton = Button(fenetre, text="Regarder dans la console", command=
    fonction_clic)
bouton.pack(padx=20, pady=20)
fenetre.mainloop()
```

Pour les bons programmeurs : il existe cependant une façon de simuler l'appel d'une fonction à paramètre lors du clic sur le bouton : c'est de créer une fonction à paramètre qui renvoie une fonction sans paramètre. C'est celle-ci qui sera passée comme argument au paramètre *command* du bouton.
Par exemple, on veut créer trois boutons qui font la même chose : afficher une couleur dans la console. Au lieu de créer trois fonctions quasiment identiques sans paramètre pour les trois boutons, on va créer une fonction qui prend un paramètre et qui renvoie une fonction sans paramètre.
C'est cette dernière qui est passée comme argument au paramètre command du bouton.

```
from tkinter import *

#Fonction à paramètre qui renvoie une fonction sans paramètre
def action_commune(couleur_a_afficher) :
    #Fonction sans paramètre qui est renvoyée par la fonction
        action_commune
    def fonction_sans_parametre() :
        print("La couleur cliquée est: ", couleur_a_afficher)
    return fonction_sans_parametre

fenetre = Tk()
#action_commune("couleur") est une fonction sans paramètre.
#Elle est passée comme argument au paramètre command du bouton
Button(fenetre, text="Rouge", command=action_commune("rouge"),
    width=10).pack(padx=10, pady=5)
Button(fenetre, text="Vert", command=action_commune("vert"), width
    =10).pack(padx=10, pady=5)
Button(fenetre, text="Bleu", command=action_commune("bleu"), width
    =10).pack(padx=10, pady=5)
fenetre.mainloop()
```

Plus d'explications :
http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/button.html

2.4 Le widget Canvas

C'est une zone rectangulaire dans laquelle on peut dessiner des formes géométriques (ligne, cercle, rectangle, polygone...), des images bitmap, du texte, ou des widgets...
On pourra par exemple utiliser ce widget pour dessiner des formes qui se déplacent ou gérer des clics de souris (voir plus loin).
Par défaut, seuls les formats d'images GIF et PGM/PPM sont lisibles par Tkinter. Mais on peut utiliser la librairie PIL (Python Imaging Library) pour lire quantité d'autres formats. Si cette librairie n'est pas installée ou si l'image n'est pas trouvée, il ne faut pas que le programme plante : on protège donc les lignes d'importation et de chargement de l'image par un try... except (voir l'exemple ci-dessous).
L'origine des coordonnées est en haut à gauche, l'axe des abscisses est vers la gauche et l'axe des ordonnées est vers le bas.

```
from tkinter import *
fenetre = Tk()
canevas = Canvas(fenetre, width=800, height=600, bg="white")
canevas.pack()
canevas.create_line(50, 50, 750, 50)
canevas.create_line(50, 100, 750, 100, fill="blue", dash=(8, 16,
    16, 8))
canevas.create_line(50, 150, 750, 150, width=5)
canevas.create_oval(50, 400, 150, 450, outline="blue", fill="yellow
    ", width=3)
canevas.create_arc(200, 400, 300, 450, outline="blue", fill="
    lightgreen", start=45, extent=90, width=3)
canevas.create_text(460, 540, font=("Comic Sans MS", 18), width
    =600, fill="grey", text="Hello")

try:
    from PIL import Image, ImageTk
    image = Image.open("linux-inside.png")
    photo = ImageTk.PhotoImage(image)
    widget = canevas.create_image(0, 600, anchor=SW, image=photo)
    canevas.tag_lower(widget)
except :
    pass
fenetre.mainloop()
```

Plus d'explications :
http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/canvas.html

3 Les animations

On veut créer une figure (par exemple une bille) qui se déplace automatiquement dans un Canvas.
Le déplacement se fait dans une fonction qui modifie les coordonnées de la bille, réaffiche la bille à sa nouvelle position et se rappelle elle-même après un certain laps de temps.
Les coordonnées de la bille doivent être déclarées comme globales afin d'être accessibles et modifiables dans la fonction qui gère le déplacement.
Il faut prévoir un test qui permet de ne plus rappeler la fonction de déplacement sous certaines conditions (par exemple si la bille atteint le bord du canevas) afin d'éviter une boucle sans fin et d'arrêter l'animation.
Le dessin qu'on veut déplacer, ici la bille qui est un ovale, doit être déclaré comme global pour être accessible dans chaque fonction de déplacement.

```
from tkinter import *
#Les coordonnées de la bille doivent être globales
#pour être accessibles et modifiées dans chaque fonction
    deplacement

x_bille, y_bille = 100, 100


def deplacement() :
    """ Déplacement de la bille """
    #On déclare les coordonnées de la bille comme globales
    global x_bille, y_bille
    #On modifie les coordonnées de la bille. L'incrément, ici 4,
       peut aussi
    #modifier la vitesse de la bille
    x_bille += 4
    y_bille += 4
    #On redessine la bille dans le canevas
    canevas.coords(bille, x_bille, y_bille, x_bille+15, y_bille+15)
    if x_bille<800 and y_bille<600 :
        #si les coordonnées sont encore dans le canevas, on
           rappelle la fonction deplacement après 15 ms
        fenetre.after(15, deplacement)

#Programme principal
fenetre = Tk()
canevas = Canvas(fenetre, width=800, height=600, bg="white")
canevas.pack()
#On donne un nom à l'item ovale
bille = canevas.create_oval(x_bille, y_bille, x_bille+15, y_bille
   +15, fill='red')
#On lance la fonction de déplacement
deplacement()
fenetre.mainloop()
```

4 Gestion du clavier

Pour utiliser les événements clavier, on utilise la méthode *bind* qui permet de relier une fonction (avec un unique paramètre nommé event) à un widget. La syntaxe est de la forme : *w.bind('<Key>', fonction_a_executer)* où *w* est un widget (par exemple un canevas), le premier paramètre est un événement clavier (ici *'<Key>'* correspond à l'appui sur n'importe quelle touche) et le deuxième paramètre est le nom de la fonction qui sera exécutée si l'événement apparaît. Le programmeur doit définir cette fonction avec un en-tête de la forme : *def fonction_a_executer(event) :*.
Il existe de nombreux événements clavier ;voir par exemple :
http://python.developpez.com/faq/?page=Evenements-Clavier#Quels-sont-les-noms-des-evenements-associes-au-clavier
Le widget doit avoir le focus pour que ça fonctionne.

```
from tkinter import *
#Nombre de pixels à chaque décalage de la bille. Permet aussi de
    régler la vitesse de la bille.
DECALAGE_PIXELS = 8
#Les décalages en abscisse et ordonnée de la bille doivent être des
     variables globales
#pour être accessibles et modifiées dans les fonctions haut, bas,
    droite, gauche
delta_x, delta_y = 0, 0
def haut(event) :
    #On déclare delta_x, delta_y comme globales pour pouvoir les
       modifier
     global delta_x, delta_y
     delta_x = 0
     delta_y = -DECALAGE_PIXELS
def bas(evente) :
    global delta_x, delta_y
    delta_x = 0
    delta_y = DECALAGE_PIXELS
def droite(event) :
    global delta_x, delta_y
    delta_x = DECALAGE_PIXELS
    delta_y = 0
def gauche(event) :
    global delta_x, delta_y
    delta_x = -DECALAGE_PIXELS
    delta_y = 0
def deplacement() :
    """ Fonction appelée régulièrement et qui redessine la bille à
       sa nouvelle position. """
    #On redessine la bille dans le canevas avec les décalages en
       abscisse et ordonnée
    canevas.move('bille', delta_x, delta_y)
    fenetre.after(30, deplacement)
#Programme principal
fenetre = Tk()
canevas = Canvas(fenetre, width=800, height=600, bg="white")
canevas.pack()
#La méthode bind permet de relier un événement à un widget (ici un
    événement clavier).
#Le widget doit avoir le focus pour que ça fonctionne.
#Le deuxième argument est une fonction à un seul paramètre (dont le
     nom est event)
canevas.bind('<Left>', gauche)
canevas.bind('<Up>', haut)
canevas.bind('<Down>', bas)
canevas.bind('<Right>', droite)
#On donne un nom à l'item ovale grâce au paramètre tag
canevas.create_oval(100, 100, 115, 115, fill='red', tag='bille')
#On donne le focus au canevas.
canevas.focus_set()
#On lance la fonction de déplacement
deplacement()
fenetre.mainloop()
```

5 Gestion de la souris

Pour utiliser les événements souris, on utilise la méthode *bind* qui permet de relier une fonction (avec un unique paramètre nommé event) à un widget lorsqu'un événement déterminé apparaît. La syntaxe

est de la forme : *w.bind('<Button-1>', fonction_a_executer)* où *w* est un widget (par exemple un canevas), le premier paramètre est un événement souris (ici *'<Button-1>'* correspond à un clic sur le bouton gauche) et le deuxième paramètre est le nom de la fonction qui sera exécutée si l'événement apparaît. Le programmeur doit définir cette fonction avec un en-tête de la forme : *def fonction_a_executer(event) :*.
Les coordonnées de la souris sont récupérables dans *event.x* et *event.y*.
Il existe de nombreux événements souris : '<Button-1>' (clic sur le bouton gauche), '<Button-3>' (clic sur le bouton droit), '<ButtonRelease-1>' (bouton gauche relâché), '<Motion>' (mouvement de la souris avec un bouton enfoncé), ...

```
from tkinter import *
from random import randint

def dessine_rond(event) :
    """ Dessine un rond à l'endroit où on clique avec le bouton
        gauche de la souris """
    rayon = randint(5, 50)
    #Les coordonnées de la souris se trouvent dans event.x et event
        .y
    canevas.create_oval(event.x-rayon, event.y-rayon, event.x+rayon
        , event.y+rayon, fill=couleurs[randint(0, 6)], width=0)

fenetre = Tk()
canevas = Canvas(fenetre, width=800, height=600, bg='white')
canevas.pack(padx=20, pady=20)
#On relie l'événement du clic sur le bouton gauche de la souris à
    la fonction dessine_rond
#La fonction dessine_rond est une fonction à un paramètre, dont le
    paramètre s'appelle event
#Ce paramètre contient les coordonnées de la souris: event.x et
    event.y
canevas.bind('<Button-1>', dessine_rond)
couleurs = ['red', 'green', 'blue', 'yellow', 'cyan', 'magenta', '
    black']
fenetre.mainloop()
```

On peut aussi relier un événement de la souris à un item (ovale, rectangle, polygone...) dessiné dans un canevas. Voici par exemple comment programmer un glisser-déposer (drag and drop en anglais) :

```
from tkinter import *
def bouton_enfonce(event):
    """ Au moment du clic sur l'item, fonction qui
    stocke les coordonnées de la souris dans old_x et old_y"""
    global old_x, old_y
    old_x = event.x
    old_y = event.y
def bouton_enfonce_deplacement(event):
    """ lorsqu'on déplace la souris en cliquant sur un bouton,
        fonction qui
    déplace l'item sur lequel on a cliqué """
    global old_x, old_y
    #On calcule le décalage entre l'ancienne position de la souris
        et la nouvelle position
    #On décale ensuite l'item de cette valeur
    #Cette méthode assure que la position relative entre l'item qu'
        on déplace et
    #le pointeur de la souris est constante
    dx = event.x - old_x
    dy = event.y - old_y
    old_x = event.x
    old_y = event.y
    canevas.move(w, dx, dy)
fenetre = Tk()
fenetre.title("Glisser-déposer")
canevas = Canvas(fenetre, width=800, height=600, bg="white")
canevas.pack(fill=BOTH, expand=YES)
canevas.create_text(400, 20, text="Cliquer sur le polygone et
    déplacer la souris en laissant le bouton enfoncé")
w = canevas.create_polygon(200, 100, 300, 250, 150, 150, fill='red'
    )
#On relie l'événement de la souris "appuyer sur le bouton gauche" à
    l'item (ici un polygone)
#dessiné dans le canevas. On exécute la fonction bouton_enfonce si
    cet événement apparaît.
canevas.tag_bind(w, "<ButtonPress-1>", bouton_enfonce)
canevas.tag_bind(w, "<B1-Motion>", bouton_enfonce_deplacement)

fenetre.mainloop()
```

6 Disposition des widgets dans leur conteneur

La disposition des widgets dans leur conteneur peut être complexe car le programme Python pourra fonctionner sur des configurations très variées (par exemple écran en 1024×768 ou en 2560× 1440) et la fenêtre pourra être redimensionnée : il faudra donc prévoir de quelle façon les widgets vont se repositionner (changement de taille, déplacement, alignement...) lors de ce redimensionnement. Il existe principalement deux méthodes en Tkinter pour positionner les widgets : la méthode *pack* et la méthode *grid*. Pour réaliser des dispositions précises, on peut imbriquer des conteneurs les uns dans les autres en utilisant le widget Frame (voir ci-dessous) mais il est déconseillé de mélanger les méthodes *pack* et *grid* dans un unique conteneur.

6.1 Méthode pack, options les plus importantes

La méthode *pack* permet de positionner des widgets les uns par rapport aux autres en spécifiant simplement s'ils sont au-dessus, en-dessous, à gauche ou à droite dans leur conteneur (Frame ou fenêtre) ou par rapport à un autre widget placé précédemment. Dès que l'interface graphique est assez compliquée, pour ne pas avoir de surprise lors de la visualisation à l'écran, ce gestionnaire nécessite l'utilisation

abondante de sous-conteneurs. Lorsqu'on utilise la méthode pack dans un conteneur, celui-ci a ensuite une taille adaptée au contenu qui a été ajouté dedans, sauf si la dimension du conteneur est imposée (de façon directe ou indirecte).
La méthode *pack* possède un certain nombre de paramètres qui permettent de préciser où un widget doit apparaître dans son conteneur et comment il doit se comporter si la fenêtre principale est redimensionnée.
Voici les principaux paramètres :

- *expand* : si le conteneur a une taille déterminée et qu'il y a beaucoup d'espace à l'intérieur, cet espace sera distribué aux widgets pour lesquels expand=1. Les widgets pour lesquels expand=0 (valeur par défaut), auront l'espace minimal qui leur convient.
- *fill* : si un widget a plus d'espace que ce qui lui convient dans le conteneur, il est possible de l'agrandir ou pas dans deux directions : en largeur (valeur X), en hauteur (valeur Y), dans les deux directions (valeur BOTH) (on peut aussi utiliser les chaînes "x", "y", "both"). Par défaut, la valeur est None (ou "none") et le widget n'est donc pas agrandi.
- *anchor* : si un widget a plus d'espace que ce qui lui convient dans le conteneur, on peut déterminer où il va se placer dans cet espace : au nord (valeur N), au nord-est (valeur NE),... La valeur par défaut est CENTER (le widget est centré horizontalement et verticalement).
- *side* : permet de disposer plusieurs widgets les uns par rapport aux autres dans le conteneur : pour les disposer horizontalement de gauche à droite, utiliser la valeur LEFT (ou "left") pour tous. Pour les disposer verticalement de haut en bas, utiliser la valeur TOP (ou "top", c'est la valeur par défaut) pour tous. De même pour BOTTOM (de bas en haut) ou RIGHT (de droite à gauche). On peut mixer par exemple les valeurs LEFT pour un widget et TOP pour un autre mais le comportement des widgets devient plus difficile à contrôler. Penser alors à utiliser la méthode *grid* à la place (voir ci-dessous).
- *ipadx*, *ipady* : crée une marge interne (horizontale ou verticale) entre le contenu du widget et le bord du widget.
- *padx*, *pady* : crée une marge externe (horizontale ou verticale) entre le bord du widget et l'extérieur du widget.

Exemple d'utilisation de la méthode *pack* :

```
from tkinter import *
fenetre = Tk()
#La fenêtre a une taille imposée
#Elle fournit donc plus d'espace aux widgets ajoutés qu'ils en ont
    besoin
fenetre.geometry('200x200+200+200')
#Pour le premier label: expand=1, et pour le deuxième: expand=0:
    donc le deuxième label
#aura uniquement l'espace vertical dont il a besoin et le deuxième
    récupérera tout le reste
#Pour le premier label, fill=None, donc il n'est pas redimensionné
    pour remplir l'espace attribué
#Pour le deuxième label, fill=X, donc il est agrandi pour occuper
    tout l'espace horizontal attribué
#Pour le premier label, anchor=E, donc il est positionné à l'est de
     l'espace attribué
#Pour le premier label, ipadx=50: 50 pixels sont ajoutés à droite
    et à gauche du contenu du label
Label(fenetre, text='Label1', bg='green').pack(expand=1, fill=None,
     anchor=E, ipadx=50)
Label(fenetre, text='Label2', bg='red').pack(expand=0, fill=X)
fenetre.mainloop()
```

6.2 Méthode grid

La méthode *grid* permet de positionner des objets directement dans une grille en spécifiant la colonne et la rangée où se situera l'objet. Cette méthode est donc utile quand l'interface graphique propose une grille de widgets avec volonté d'avoir des alignements respectés. Il faut éviter, dans un même conteneur, d'utiliser à la fois la méthode *grid* et la méthode *pack*.
Voici les principaux paramètres de la méthode *grid* :

- *row* : entier (0 par défaut) qui spécifie le numéro de ligne où doit se trouver le widget.
- *column* : entier (0 par défaut) qui spécifie le numéro de colonne où doit se trouver le widget.
- *columnspan* : permet de fusionner plusieurs cellules d'une colonne : entier qui spécifie le nombre de cellules à fusionner verticalement.
- *rowspan* : permet de fusionner plusieurs cellules d'une ligne : entier qui spécifie le nombre de cellules à fusionner horizontalement.
- *sticky* : si un widget a plus d'espace que ce qui lui convient dans la cellule, ce paramètre permet de préciser comment on l'aligne dans cette cellule, ou bien comment on l'agrandit pour qu'il remplisse la cellule. Par défaut , le widget n'est pas agrandi et centré horizontalement et verticalement.
  sticky=NE n'agrandit pas le widget et le place en haut à droite de la cellule.
  sticky=W n'agrandit pas le widget et le place à gauche de la cellule, en le centrant verticalement.
  sticky=E+W agrandit le widget horizontalement mais pas verticalement (il est centré verticalement).
  sticky=S+N agrandit le widget verticalement mais pas horizontalement (il est centré horizontalement).
  sticky=S+N+E+W agrandit le widget , qui remplit alors toute la cellule.
  sticky=E+W+N agrandit le widget horizontalement mais pas verticalement (il est placé en haut de la cellule).
- *ipadx*, *ipady* : crée une marge interne (horizontale ou verticale) entre le contenu du widget et le bord du widget.
- *padx*, *pady* : crée une marge externe (horizontale ou verticale) entre le bord du widget et l'extérieur du widget.

```
from tkinter import *
fenetre = Tk()
#Empêche la fenêtre d'être redimensionnée
fenetre.resizable(width=False, height=False)
texte1 = Label(fenetre, text = 'Premier champ :')
texte2 = Label(fenetre, text = 'Second :')
entree1 = Entry(fenetre)
entree2 = Entry(fenetre)
#texte1 est affiché dans la cellule (0,0) et aligné à gauche
texte1.grid(row =0, sticky=W, padx=5)
#texte1 est affiché dans la cellule (1,0) et aligné à gauche
texte2.grid(row =1, sticky=W, padx=5)
#entree1 est affiché dans la cellule (0,1) avec une
#marge externe horizontale et verticale
entree1.grid(row =0, column =1, padx=5, pady=5)
#entree2 est affiché dans la cellule (1,1) avec une
#marge externe horizontale et verticale
entree2.grid(row =1, column =1, padx=5, pady=5)
fenetre.mainloop()
```

6.3 Widget Frame

Le widget Frame est utilisé principalement comme un conteneur d'autres widgets. Il permet d'ajouter des possibilités pour disposer les widgets dans une fenêtre.

7 Liens utiles

- http://effbot.org/tkinterbook/
- http://www.jchr.be/python/tkinter.htm
- https://docs.python.org/3/library/tkinter.html
- http://fr.wikibooks.org/wiki/Programmation_Python/Tkinter
- http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html