Table des matières

Chapitre 1

Interfaces graphiques en Python avec Tkinter

1.1 Les interfaces graphiques en Python

Python fournit diverses options pour développer des interfaces **graphiques GUI** :

1. **Tkinter :** Tkinter est l'interface Python de la bibliothèque GUI Tk livrée avec Python. Nous allons l'étudier en détail sur ce chapitre.
2. **wxPython :** Ceci est une implémentation en Python libre et open source Python de l'interface de programmation wxWidgets.
3. **PyQt :** Il s'agit également d'une interface Python pour une bibliothèque d'interface graphique Qt populaire multiplate-forme.
4. **JPython :** JPython est un outil Python pour Java, qui donne aux scripts Python un accès transparent aux bibliothèques de classes Java.

Il existe de nombreuses autres interfaces graphiques disponibles, que vous pouvez trouver sur le net.

1.2 La bibliothèque graphique Tkinter

Tkinter est la bibliothèque d'**interface graphique standard** pour **Python**. Python, lorsqu'il est combiné à **Tkinter**, fournit un moyen rapide et facile pour créer des applications graphiques. **Tkinter** fournit une puissante interface orientée objet simple et conviviale.

La création d'une application graphique à l'aide de Tkinter est une tâche assez facile. Tout ce que vous avez à faire est de suivre les étapes suivantes :

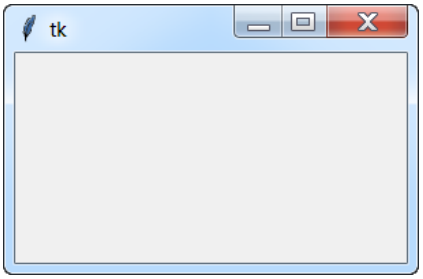
1. ***Importez le module Tkinter.***
2. ***Créez la fenêtre principale de l'application graphique.***
3. ***Ajoutez un ou plusieurs des widgets graphiques.***

4. ***Entrez la boucle d'évènements principale pour agir contre chaque évènement déclenché par l'utilisateur.***

Exemple. création d'une simple fenetre Tkinter

```
# -*- coding : utf-8 -*-
# Importation de la bibliothèque tkinter
from tkinter import*

# Création d'une fenêtre tkinter
maFenetre = Tk()

# vos widgets ici : bouton de commande, champ de saisie , labels ...

# Entrez la boucle d'événements principale
maFenetre.mainloop()
```

Ce qui affiche après exécution :



1.3 Les widgets Tkinter

La bibliothèque Tkinter fournit divers contrôles, tels que des boutons, des étiquettes et des zones de texte utilisées dans une application graphique. Ces contrôles sont communément appelés **widgets.**

Il existe actuellement **15 types de widgets dans Tkinter.** Nous présentons ici les noms de ces widgets ainsi qu'une brève description :

1. **Button :** le widget Button permet de créer des boutons pour votre application.
2. **Canva :** le widget Canva permet de dessiner des formes, telles que des lignes, des ovales, des polygones et des rectangles, dans votre application.
3. **Checkbutton :** le widget Checkbutton permet d'afficher un certain nombre d'options sous forme de cases à cocher. L'utilisateur peut sélectionner plusieurs options à la fois.
4. **Entry :** le widget Entry est utilisé pour afficher un champ de texte d'une seule ligne permettant d'accepter les valeurs d'un utilisateur.
5. **Frame :** le widget Frame (cadre) est utilisé en tant que widget conteneur pour organiser d'autres widgets.
6. **Label :** le widget Label est utilisé pour fournir une légende ou description pour les autres widgets. Il peut aussi contenir des images.

7. **Listbox :** le widget Listbox est utilisé pour fournir une liste d'options à un utilisateur.
8. **menubutton :** le widget menubutton est utilisé pour afficher les menus dans votre application.
9. **Menu :** le widget Menu est utilisé pour fournir diverses commandes à un utilisateur. Ces commandes sont contenues dans Menubutton.
10. **Message :** le widget Message est utilisé pour afficher des champs de texte multilignes permettant d'accepter les valeurs d'un utilisateur.
11. **Radiobutton :** le widget Radiobutton est utilisé pour afficher un certain nombre d'options sous forme de boutons radio. L'utilisateur ne peut sélectionner qu'une option à la fois.
12. **Scale :** le widget Echelle est utilisé pour fournir un widget à curseur.
13. **Scrollbar :** le widget Scrollbar ou barre de défilement est utilisé pour ajouter une fonctionnalité de défilement à divers widgets, tels que les zones de liste.
14. **Text :** le widget Text est utilisé pour afficher du texte sur plusieurs lignes.
15. **Toplevel :** le widget Toplevel est utilisé pour fournir un conteneur de fenêtre séparé.
16. **Spinbox :** le widget Spinbox est une variante du widget standard **Tkinter Entry**, qui peut être utilisé pour sélectionner un nombre fixe de valeurs.
17. **PanedWindow :** le widget PanedWindow est un conteneur pouvant contenir un nombre quelconque de volets, disposés horizontalement ou verticalement.
18. **LabelFrame :** un labelframe est un simple widget de conteneur. Son objectif principal est d'agir comme un intercalaire ou un conteneur pour les dispositions de fenêtre complexes.
19. **tkMessageBox :** ce module est utilisé pour afficher des boîtes de message dans vos applications.

1.4 Exemple de widget Tkinter

1.4.1 Le widget Button

Pour créer un **bouton de commande** sur une fenêtre Tkinter, on utilise la syntaxe :

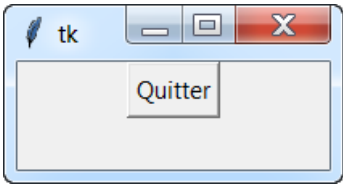
```
Nom_du_bouton = Button( Nom_de_la_fenêtre , text =" Texte du bouton" , action =
    action_du_bouton() )
```

Exemple. bouton quiter l'application

```
# -*- coding : utf-8 -*-
from tkinter import*
maFenetre = Tk()

#Création d'un bouton de commande
b = Button(maFenetre , text = "Quitter" , command = quit)

```

```
# Placer le bouton sur la fenêtre
b.pack()

maFenetre.mainloop()
```

Ce qui affiche à l'exécution :



Avec l'action : "quitter la fenêtre en cliquant sur le bouton"

1.4.2 Le widget label

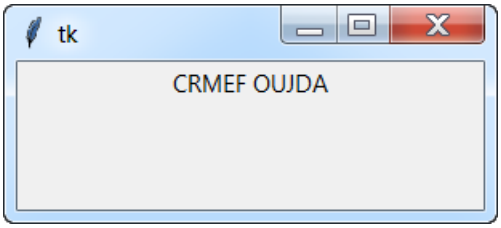
L'insertion d'un lebel sur une fenêtre Tkinter est semblable à celui d'un bouton de commande :

```
Nom_du_label = Label( Nom_de_la_fenêtre, text ="Texte du label")
```

Exemple. label

```
# -*- coding : utf-8 -*-
from tkinter import*
maFenetre = Tk()

#Création du label
lbl = Label(maFenetre , text = "CRMEF OUJDA")
# Placer le label sur la fenêtre
lbl.pack()

maFenetre.mainloop()
```

Ce qui affiche après exécution :



Remarque 1.4.1. On peut aussi utiliser un texte variable, afin de donner la possibilité à l'utilisateur de modifier le texte :

Exemple. texte variable

```
# -*- coding : utf-8 -*-
from tkinter import *
fen = Tk()
var = StringVar()
```

```
label = Label( fen, textvariable=var)
var.set("CRMEF OUJDA")
label.pack()
fen.mainloop()
```

On peut aussi changer le texte via une action associé à un bouton de commande

Exemple. ...

```
# -*- coding : utf-8 -*-
from tkinter import *
fen = Tk()
def action () :
    var.set("J'ai changé le texte en cliquant")
var = StringVar()
label = Label( fen, textvariable=var)
var.set("CRMEF OUJDA")
b = Button(fen, text = "Essayez", command=action)
b.pack()
label.pack()
fen.mainloop()
```

1.4.3 Le champ de saisie Entry

1.4.3.1 Définition et syntaxe générale du widget Entry

Le champ Entry permet de créer une zone de saisie sur une seule ligne, pouvant servir à la récupération des données saisie par l'utilisateur afin les utiliser comme variables (formulaire d'inscription, formulaire d'identification...). La syntaxe est :

```
Nom_du_champ = Entry( Nom_de_la_fenêtre,options...)
```

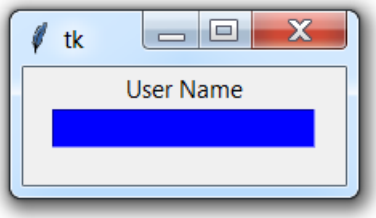
Syntaxe Voici la syntaxe pour créer un widget Entry

```
w = Entry (master, option, ...)
# master : fenêtre parente.
#options : la liste des options pour le widget Entry.
```

Exemple. Widget Entry

```
from tkinter import *
root = Tk()
user_Entry = Entry(root,bg="blue")
user = Label(root, text = "User Name")
user.pack()
user_Entry.pack()
root.mainloop()
```

Ce qui affiche après exécution :



1.4.3.2 Liste des options pour le widget Entry

1. **bg :** définit la couleur d'arrière-plan du widget Entry.
2. **bd :** définit la taille des bordures. La valeur par défaut est 2 pixels.
3. **command :** définit et associe une commande
4. **cursor :** définit le motif du curseur de la souris
5. **font :** définit la police qui sera utilisée pour le texte.
6. **exportselection :** par défaut, si vous sélectionnez du texte dans un widget Entry, il peut être exporté vers le Presse-papiers. Pour éviter cette exportation, utilisez export-selection = 0.
7. **fg :** définit la couleur qui sera utilisée pour le texte
8. **highlightcolor :** Il représente la couleur de surbrillance à utiliser pour le rectangle qui est dessiné autour du widget lorsqu'il a le focus.
9. **justify :** Il spécifie comment le texte est organisé s'il contient plusieurs lignes. 10relief Spécifie le type de bordure. Sa valeur par défaut est FLAT.
10. **selectbackground :** la couleur d'arrière-plan du texte sélectionné.
11. **selectborderwidth :** définit la largeur de la bordure à utiliser autour du texte sélectionné. La valeur par défaut est un pixel.
12. **selectforeground :** définit la couleur de premier plan du texte sélectionné.
13. **show :** est utilisé pour afficher ou masquer les caractères. Exemple pour un mot de passe on utilise show = "*".
14. **state :** la valeur par défaut est state = NORMAL, mais vous pouvez utiliser state = DISABLED pour masquer le contrôle et le rendre inactif.
15. **textvariable :** Pour pouvoir extraire le texte actuel de votre widget Entry, vous devez définir cette option sur une instance de la classe StringVar.
16. **width :** définit la largeur du widget
17. **xscrollcommand :** ajoute une barre de défilement au widget.

1.4.3.3 Les méthodes associés au widget Entry

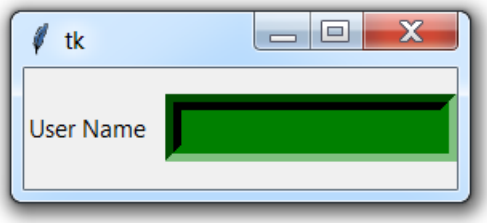
1. **delete :** (first, last = None) : Supprime les caractères du widget, en commençant par celui qui est à l'index first, jusqu'au dernier last

2. **get() :** renvoie le texte actuel de l'entrée sous forme de variable chaîne.
3. **icursor(index) :** place le curseur d'insertion juste avant le caractère à l'index donné.
4. **index(index) :** décale le contenu de l'entrée de sorte que le caractère à l'index donné soit le caractère visible le plus à gauche. N'a aucun effet si le texte est entièrement contenu dans l'entrée.
5. **insert(index, s) :** insère la chaîne s avant le caractère à l'index donné.
6. **select_adjust (index) :** cette méthode permet de s'assurer que la sélection inclut le caractère à l'index spécifié.
7. **select_clear () :** efface la sélection. S'il n'y a pas actuellement de sélection, n'a aucun effet.
8. **select_form(index) :** définit la position de l'index d'ancrage sur le caractère spécifié par l'index.
9. **select_present() :** s'il y a une sélection, renvoie vrai, sinon renvoie faux.
10. **select_range(début, fin) :** sélectionne les caractères qui existent dans la plage spécifiée.
11. **select_to(index) :** sélectionne tous les caractères du début à l'index spécifié.
12. **xview(index) :** utilisé pour lier le widget de saisie à une barre de défilement horizontale.
13. **xview_scroll() :** utilisé pour faire défiler l'entrée horizontalement.

Exemple. widget Entry

```
1 from tkinter import *
2  root = Tk()
3 user = Label(root, text = "User Name")
4  user.pack( side = LEFT )
5  user_Entry = Entry(root, bd = 10, bg="green")
6 user_Entry.pack( side=RIGHT )
7  root.mainloop()
```

Ce qui affiche après exécution :



1.4.4 Associer une action ou évènement à un Widget Tkinter

1.4.4.1 Associer une action à un bouton de commande

1.4.4.2 Associer un évènement à un bouton de commande (bind action)

1.4.4.3 Associer un évènement à un champ Entry (bind action)

On peut associer un évènement à un champ de saisi à l'aide de la méthode **bind()** :

```
entry.bind("<Return>", actionEvent)
```

Exemple. Evènement associé à un champ de saisi :

```
def actionEvent(event):
    lbl.configure(text = "Vous avez tapé = "+ entry.get())

root = Tk()
entry = Entry(root)

# Association de lévènement actionEvent au champ de saisie
entry.bind("<Return>", actionEvent)
lbl = Label(root, text = ".....")
entry.pack()
lbl.pack()
root.mainloop()
```

1.4.5 Le widget Text

1.4.5.1 Syntaxe & description du widget Text

Le widget Text offre des fonctionnalités avancées vous permettant d'éditer un texte multilingue et de formater son affichage, en modifiant sa couleur et sa police...

Vous pouvez également utiliser des structures élégantes telles que des onglets et des marques pour localiser des sections spécifiques du texte et appliquer des modifications à ces zones. De plus, vous pouvez incorporer des fenêtres et des images dans le texte car ce **widget** a été conçu pour gérer à la fois le texte brut et le texte mis en forme.

Syntaxe pour du widget **Text**

```
Nom_du_widget_Text = Texte (parent, options, ...)
```

Le paramètre parent : représente la fenêtre parent.
options : représente la liste des options pour le widget.

1.4.5.2 Les options disponibles pour le widget Text

Voici la liste des options les plus utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules :

1. **bg :** la couleur d'arrière-plan par défaut du widget de texte.
2. **bd :** la largeur de la bordure autour du widget de texte. La valeur par défaut est 2 pixels.

3. **cursor :** le curseur qui apparaîtra lorsque la souris survolera le widget texte.
4. **exportselection :** normalement, le texte sélectionné dans un widget de texte est exporté pour être sélectionné dans le gestionnaire de fenêtres. Définissez exportselection = 0 si vous ne voulez pas ce comportement.
5. **font :** la police par défaut pour le texte inséré dans le widget.
6. **fg :** la couleur utilisée pour le texte (et les bitmaps) dans le widget. Vous pouvez changer la couleur pour les régions marquées ; cette option est juste la valeur par défaut.
7. **height :** la hauteur du widget en lignes (pas en pixels !), Mesurée en fonction de la taille de la police actuelle.
8. **highlightbackbackground :** la couleur du focus est mise en surbrillance lorsque le widget de texte n'a pas le focus.
9. **highlightcolor :** la couleur du focus est mise en surbrillance lorsque le widget de texte a le focus.
10. **highlightthickness :** l'épaisseur du focus est mise en évidence. La valeur par défaut est 1. Définissez l'épaisseur de la sélection = 0 pour supprimer l'affichage de la surbrillance.
11. **insertbackground :** la couleur du curseur d'insertion. Le défaut est noir.
12. **insertborderwidth :** taille de la bordure 3D autour du curseur d'insertion. La valeur par défaut est 0.
13. **insertofftime :** le nombre de millisecondes pendant lequel le curseur d'insertion est désactivé pendant son cycle de clignotement. Définissez cette option sur zéro pour supprimer le clignotement. La valeur par défaut est 300.
14. **insertontime :** le nombre de millisecondes pendant lequel le curseur d'insertion est activé pendant son cycle de clignotement. La valeur par défaut est 600.
15. **insertwidth :** largeur du curseur d'insertion (sa hauteur est déterminée par l'élément le plus grand de sa ligne). La valeur par défaut est 2 pixels.
16. **padx :** la taille du remplissage interne ajouté à gauche et à droite de la zone de texte. La valeur par défaut est un pixel.
17. **pady :** la taille du remplissage interne ajouté au-dessus et au-dessous de la zone de texte. La valeur par défaut est un pixel.
18. **relief :** l'apparence 3-D du widget de texte. La valeur par défaut est relief = SUNKEN.
19. **selectbackground :** la couleur de fond à utiliser pour afficher le texte sélectionné.
20. **selectborderwidth :** la largeur de la bordure à utiliser autour du texte sélectionné.
21. **spacing1 :** cette option spécifie combien d'espace vertical supplémentaire est placé au-dessus de chaque ligne de texte. Si une ligne est renvoyée à la ligne, cet espace est ajouté uniquement avant la première ligne occupée à l'écran. La valeur par défaut est 0.
22. **spacing2 :** cette option spécifie la quantité d'espace vertical supplémentaire à ajouter entre les lignes de texte affichées lorsqu'une ligne logique est renvoyée à la ligne. La valeur par défaut est 0

23. **spacing3 :** cette option spécifie combien d'espace vertical supplémentaire est ajouté en dessous de chaque ligne de texte. Si une ligne est renvoyée à la ligne, cet espace est ajouté uniquement après la dernière ligne occupée à l'écran. La valeur par défaut est 0.
24. **state :** normalement, les widgets de texte répondent aux évènements de clavier et de souris ; set state = NORMAL pour obtenir ce comportement. Si vous définissez state = DISABLED, le widget texte ne répondra pas et vous ne pourrez pas non plus modifier son contenu par programme.
25. **tabs :** cette option contrôle la manière dont les caractères de tabulation positionnent le texte.
26. **width :** la largeur du widget en caractères (pas en pixels !), Mesurée en fonction de la taille de la police actuelle.
27. **wrap :** cette option contrôle l'affichage des lignes trop larges. Définissez wrap = WORD et la ligne sera coupée après le dernier mot qui convient. Avec le comportement par défaut, wrap = CHAR, toute ligne trop longue sera brisée par n'importe quel caractère.
28. **xscroll :** pour que le widget texte puisse défiler horizontalement, définissez cette option sur la méthode **set()** de la barre de défilement horizontale.
29. **yscroll :** pour que le widget texte puisse défiler verticalement, définissez cette option sur la méthode set() de la barre de défilement verticale.

1.4.5.3 Les méthodes associées au widget Text

Voici la liste des principales **méthodes** associées à l'objet **Text**

1. **delete (startindex [, endindex]) :** cette méthode supprime un caractère spécifique ou une plage de texte.
2. **get (startindex [, endindex])** : cette méthode retourne un caractère spécifique ou une plage de texte.
3. **index (index) :** etourne la valeur absolue d'un index basé sur l'index donné.
4. **insert (index [, string] ...) :** cette méthode insère des chaînes à l'emplacement d'index spécifié.
5. **see(index) :** ette méthode retourne true si le texte situé à la position d'index est visible.
6. **mark_gravity (mark [, gravity]) :** retourne la gravité de la marque donnée. Si le deuxième argument est fourni, la gravité est définie pour la marque donnée.
7. **mark_names () :** retourne toutes les marques du widget Texte.
8. **mark_set (mark, index) :** informe une nouvelle position par rapport à la marque donnée.
9. **mark_unset (mark) :** supprime la marque donnée du widget Texte.
10. **tag_add (tagname, startindex [, endindex] ...) :** cette méthode balise la position définie par startindex ou une plage délimitée par les positions startindex et endindex.

11. **tag_config() :** Vous pouvez utiliser cette méthode pour configurer les propriétés de la balise, qui comprennent, justifier (centre, gauche ou droite), des onglets (cette propriété a les mêmes fonctionnalités que la propriété des onglets du widget Texte) et un soulignement (utilisé pour souligner le texte marqué).).
12. **tag_delete (tagname) :** cette méthode est utilisée pour supprimer une balise donnée.
13. **tag_remove (tagname [, startindex [.endindex]] ...) :** après avoir appliqué cette méthode, la balise donnée est supprimée de la zone fournie sans supprimer la définition de balise réelle.

Exemple. Tkinter Text Widget

```
# -*- coding : utf-8 -*-
from tkinter import *
root= Tk()
t = Text(root, fg="red", padx=50)
t.config(font=("broadway", 14))
t.insert(1.0, "Exemple de widget Text Tkinter ")
t.pack()
root.mainloop()
```

Ce qui affiche après exécution :

1.4.6 Le widget Frame Tkinter

Le widget **Frame (cadre)** est très important pour le processus de regroupement et d'organisation des autres widgets de manière conviviale. Cela fonctionne comme un conteneur, qui est responsable de la position des autres widgets.

Il utilise des zones rectangulaires à l'écran pour organiser la mise en page et fournir un remplissage de ces widgets. Un **Frame** peut également être utilisé comme classe de base pour implémenter des widgets complexes.

1.4.6.1 Syntaxe

```
w = Frame (master, option, ...)
```

1. **master** : représente la fenêtre parente.
2. **options :** liste des options les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

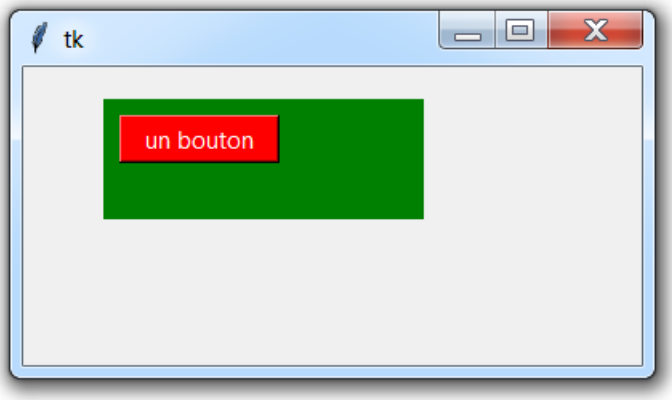
1.4.6.2 Liste des options d'un widget Frame

1. **bg :** couleur d'arrière-plan du widget Frame
2. **bd :** taille des bordures autour du Frame.
3. **cursor :** permet de personnaliser le motif du curseur de la souris au moment du survole.
4. **height :** définit la dimension verticale du Frame.
5. **highlightbackground :** définit la couleur de la mise au point en surbrillance de l'objet Frame.
6. **highlightcolor :**
7. **relief :** avec la valeur par défaut **relief = FLAT**, la case à cocher ne ressort pas de son arrière-plan. Vous pouvez définir cette option sur n'importe quel autre style.
8. **width :** définit la largeur du frame

Exemple. Frame Tkinter

```
1  # -*- coding : utf-8 -*-
2  from tkinter import *
3
4  # Création de la fenêtre principale
5  master = Tk()
6  master.geometry("400x200")
7
8  #Création d'un frame d'arrière plan vert
9  frm = Frame(master, bg='green')
10
11 # Emplacement du frame
12  frm.place(x=50, y=20, width=200,height=75)
13
14 # Création d'un bouton au sein du frame
15 b = Button(frm, text="un bouton",bg='red', fg='white')
16  b.place(x=10, y=10, width=100,height=30)
17 master.mainloop()
```

Ce qui affiche après exécution :



1.5 Les attributs standard des widgets Tkinter

Malgré la diversité de leurs méthodes et propriétés, les widgets Tkinter possèdent des attributs communs, tels que la taille, la couleur et la police sont spécifiés :

1. **Dimensions**
2. **Colors**
3. **Fonts**
4. **Anchors**
5. **Relief styles**
6. **Bitmaps**
7. **Cursors**

Exemple. Label avec couleur de font et background personnalisés

```
1 # -*- coding : utf-8 -*-
2 from tkinter import *
3 #Création d'une fenêtre Tkinter
4 f = Tk()
5 f.geometry("300x75")
6 #Création d'un  widget label de couleur blanche, bold, taille 18...
7 Centre = Label(f, text = "CRMEF OUJDA ", bg="black", fg="white", font='broadway 18 bold')
8  Centre.pack()
9 f.mainloop()
```

Ce qui affiche après exécution :

1.6 Les méthodes de gestion des dispositions géométriques des widgets

Tous les **widgets Tkinter** ont accès aux méthodes de gestion de géométrie spécifiques, qui ont pour but d'organiser les widgets dans la zone du widget parent. Tkinter possède les classes de gestionnaire de géométrie suivantes :

1. **La méthode pack()**
2. **La méthode grid()**
3. **La méthode place()**

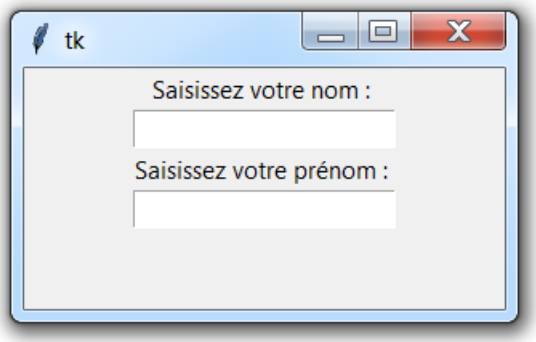
1.6.1 La méthode de disposition géométrique pack()

Le gestionnaire de disposition géométrique **pack()**, organise les widgets en blocs avant de les placer dans le widget parent. Les widgets sont placés l'un au dessous de l'autre selon l'ordre d'application de la méthode pack() avec un emplacement **centré par défaut.**

Exemple. usage de la méthode **pack()**

```
1  # -*- coding : utf-8 -*-
2  from tkinter import *
3
4  #Création d'une fenêtre Tkinter
5  f = Tk()
6  f.geometry("300x150")
7  Nom = Label(f, text = "Saisissez votre nom : ")
8  Prenom = Label(f, text="Saisissez votre prénom : ")
9  champNom = Entry(f) champPrenom = Entry(f)
10 Nom.pack()
11 champNom.pack()
12 Prenom.pack()
13 champPrenom.pack()
14 f.mainloop()
```

Ce qui affiche après exécution :



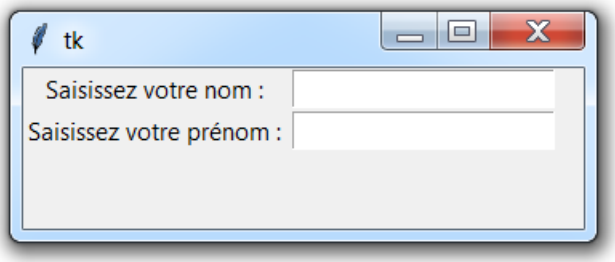
1.6.2 La méthode de disposition géométrique grid()

Le gestionnaire de disposition géométrique **grid()**, organise les widgets dans une structure de type table dans le widget parent.

Exemple. usage de la méthode **grid()**

```
# -*- coding : utf-8 -*-
from tkinter import *

#Création d'une fenêtre
Tkinter f = Tk()
f.geometry("350x100")

#Création des widgets
Nom = Label(f, text = "Saisissez votre nom : ")
Prenom = Label(f, text="Saisissez votre prénom : ")
champNom = Entry(f)
champPrenom = Entry(f)

#Application de la méthode grid() aux widget
Nom.grid(row=0, column=0)
Prenom.grid(row=1, column=0)
champNom.grid(row=0, column=1)
champPrenom.grid(row=1, column=1)
f.mainloop()
```

Ce qui affiche après exécution :



1.6.3 La méthode de disposition géométrique place()

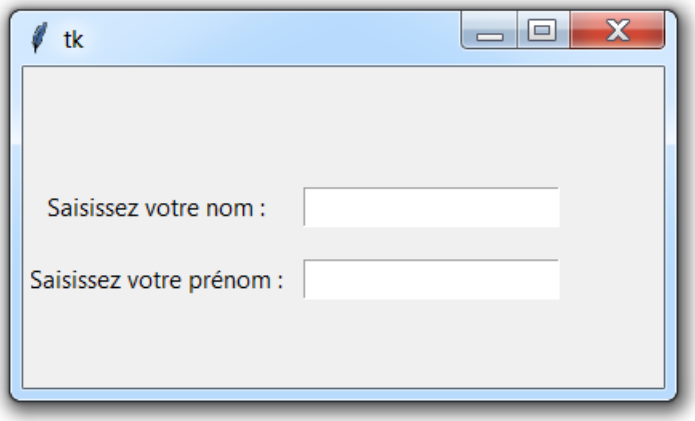
Le gestionnaire de disposition géométrique **place()**, organise les widgets en les plaçant à des positions spécifiques dans le widget parent suivant leurs coordonnées et leurs dimensions :

Exemple. Usage de la méthode **place() :**

```
# -*- coding : utf-8 -*-
from tkinter import *

#Création d'une fenêtre Tkinter
f = Tk()
f.geometry("400x200")

#Création des widgets
```

```
Nom = Label(f, text = "Saisissez votre nom : ")
Prenom = Label(f, text="Saisissez votre prénom : ")
champNom = Entry(f)
champPrenom = Entry(f)
#Application de la méthode place() aux widget
Nom.place(x=5, y=75, width=160, height=25)
champNom.place(x=175, y=75, width=160, height=25)
Prenom.place(x=5, y=120, width=160, height=25)
champPrenom.place(x=175, y=120, width=160, height=25)
f.mainloop()
```

Ce qui affiche après exécution :



1.7 Actions manipulant des widgets Tkinter

1.7.1 Action associée à un bouton de commande

1.8 Menu Tkinter en Python

Le rôle de ce widget est de vous permettre de créer toutes sortes de menus utilisables par vos applications. La création d'un menu se déroule selon les étapes suivantes :

1. **Création de la barre des menus**

```
menuBar = Menu (master) # master designe la fenêtre principale
```

2. **Création d'un menu principale :**

```
menuPrincipal = Menu(menuBar, tearoff = 0)
#tearoff = 0 pour menu non détachable, tearoff=1 pour menu détachable
menuBar.add_cascade(label = "label du menu principal", menu=menuPrincipal)
```

3. **Création d'une commande ou sous menu du menu principal**

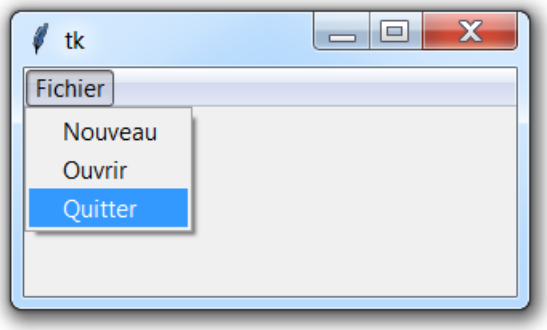
```
menuPrincipal.add_command(label="Sous_menu", command = "..")
```

4. Configuration de la barre des menus

```
master.config(menu = menuBar)
```

Exemple. menu Fichier

```
# -*- coding : utf-8 -*-
from tkinter import *
master = Tk()

# Création de la barre des menu
menuBar = Menu(master)

# Création du menu principal 'Fichier'
menuFichier  = Menu(menuBar, tearoff = 0)
menuBar.add_cascade(label="Fichier",menu=menuFichier)

# Création des sous menus : 'Nouveau', 'Ouvrir', 'Quitter'
menuFichier.add_command(label="Nouveau")
 menuFichier.add_command(label="Ouvrir")
menuFichier.add_command(label="Quitter", command=quit)

# Configuration de la barre des menus
master.config(menu=menuBar)
master.mainloop()
```



Ce qui affiche après exécution :