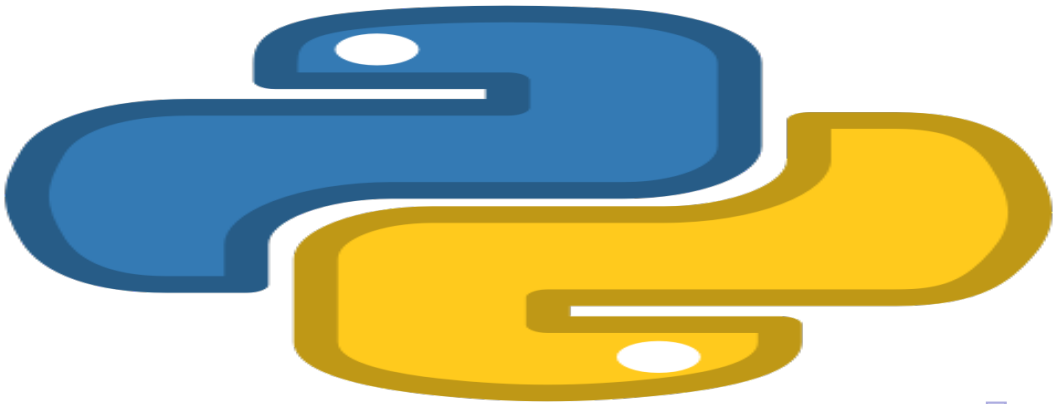
Python : tkinter

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

1. Introduction
2. Fenêtre principale
   - Titre
   - Dimension
3. Widget de base
   - `Button`
   - `messagebox`
   - `Label`
   - `Entry`
   - `Listbox`
   - `Spinbox`
   - `Menu`
   - `Text`

Plan

4. Widget ttk
   - `Combobox`
   - `Treeview`
   - `Progressbar`
5. Fermeture d'un frame ou d'un widget
6. Évènements
7. Validation d'un champ

Plan

8 Multi-frames
- Remplacer les frames
- Afficher plusieurs frames
- Afficher une application web dans un frame
- Ajouter un lien vers une page web

9 Frames réutilisables
- Solution avec les fonctions
- Solution avec les classes

Python

tkinter

- Ou **Tkinter** dans la version 2 de **Python**
- Bibliothèque graphique pour **Python** : permettant de créer des interfaces graphiques (UI) ou des applications Desktop
- Intégré à la bibliothèque standard de **Python**
- Offrant des widgets (éléments d'interface utilisateur) : boutons, labels, fenêtres...
- Compatible avec plusieurs systèmes d'exploitation : **Windows**, **macOS** et **Linux**
- Bibliothèque graphique la plus utilisée en **Python**

Python

Tkinter, **Tk** et **Tcl**

- **Tcl** (**T**ool **C**ommand **L**anguage)
  - un langage de script créé par **John Ousterhout** en 1988,
  - peut être utilisé indépendamment de **Python**.
- **Tk** (**T**ool**K**it)
  - est une bibliothèque graphique créée par **John Ousterhout** en 1990,
  - associée au langage **Tcl**.
- **Tkinter** est l'interface qui relie **Python** à **Tcl/Tk** pour créer des applications graphiques.

Python

Autres bibliothèques graphiques pour **Python**

- **PyQT**
- **Kivy**
- **WxPython**
- **PySide**
- **...**

Python

tkinter : avantages

- Inclus dans la bibliothèque standard de Python
- Simplicité d'utilisation
- Multiplateforme
- Widgets prêts à l'emploi
- Documentation étendue

tkinter : inconvénients

- Apparence basique
- Performance
- Écosystème externe limité

ASP.NET Core

Pour une première fenêtre de démonstration Tkinter, exécutez

```
python -m tkinter
```

Python

Avant de commencer

- Créez un nouveau projet `cours_tkinter` dans votre espace de travail
- Créez un fichier `main.py`
- Validez

Python

Avant de commencer

- Créez un nouveau projet `cours_tkinter` dans votre espace de travail
- Créez un fichier `main.py`
- Validez

À consulter avant d'utiliser **Tkinter** sous **Mac**

`https://stackoverflow.com/questions/63381847/install-tkinter-on-mac-os-and-pyenv`

Python

Remarques

- Il existe des problèmes d'incompatibilité entre **Python** 3.13 et **Tkinter**, en particulier liés à l'absence du fichier `init.tcl`, qui est essentiel pour le fonctionnement de **Tkinter**
- Rétrograder vers **Python** 3.12 pourrait être une solution temporaire.

Python

Commençons par importer le module

```
from tkinter import Tk
```

Python

Commençons par importer le module

```
from tkinter import Tk
```

Pour créer une fenêtre principale

```
root = Tk()
```

Python

Commençons par importer le module

```
from tkinter import Tk
```

Pour créer une fenêtre principale

```
root = Tk()
```


Et enfin, une méthode pour maintenir la fenêtre affichée (grâce à une boucle infinie)

```
root.mainloop()
```

Python

Pour attribuer à un titre à la fenêtre

```
from tkinter import Tk

root = Tk()
root.title("First frame")

root.mainloop()
```

Python

Pour changer sa dimension (largeur :600px, hauteur : 300px)

```
from tkinter import Tk

root = Tk()
root.title("First frame")
root.geometry("600x300")

root.mainloop()
```

Python

On peut évidemment passer les paramètres dans une chaîne formatée

```
from tkinter import Tk

root = Tk()
root.title("First frame")

width = 600
height = 300

root.geometry(f"{width}x{height}")

root.mainloop()
```

Python

La méthode `geometry` peut prendre comme paramètre les coordonnées du point haut gauche

```
from tkinter import Tk

root = Tk()
root.title("First frame")

width = 600
height = 300
x = 500
y = 500

root.geometry(f"{width}x{height}+{x}+{y}")

root.mainloop()
```

Python

Pour que le frame soit affiché en plein écran

```
from tkinter import Tk

root = Tk()
root.title("First frame")

width = root.winfo_screenwidth()
height = root.winfo_screenheight()

root.geometry(f"{width}x{height}")

root.mainloop()
```

Python

Exercice

- Dans un nouveau fichier, créer un nouveau frame
- Ses dimensions : 500 (largeur) et 300 (hauteur)
- Le frame doit être affiché au centre de l'écran

Python

Correction

```
from tkinter import Tk

root = Tk()
root.title("First frame")

screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

width = 500
height = 300

x = (screen_width - width)//2
y = (screen_height - height)//2

root.geometry(f"{width}x{height}+{x}+{y}")

root.mainloop()
```

Python

Quelques widgets de base

- **Button**
- **Label**
- **Entry** (zone de saisie multi-ligne)
- **Text** (zone de saisie mono-ligne)
- **Checkbutton**
- **Radiobutton**
- **Menu**
- ...

Python

Remarque

- Les propriétés d'un widgets peuvent être spécifiées à l'instanciation
- Et elles peuvent être modifiées en appelant la méthode `config`

Python

Avant de commencer

- Pour chaque widget, nous créerons un fichier séparé
- La restructuration du projet sera abordée dans la dernière section de ce chapitre

Python

Commençons par importer `Button`

```
from tkinter import Tk, Button
```

Python

Commençons par importer `Button`

```
from tkinter import Tk, Button
```

Pour créer un bouton

```
button = Button(root, text='Afficher')
```

Python

Commençons par importer `Button`

```
from tkinter import Tk, Button
```

Pour créer un bouton

```
button = Button(root, text='Afficher')
```


Explication

- Le bouton a été associé au frame
- Le texte `Afficher` sera marqué sur le bouton

Python

Quelques autres paramètres du constructeur `Button`

- `bg` : pour modifier la couleur du fond du bouton.
- `font` : pour modifier le font du texte marqué sur le bouton.
- `image` : pour afficher une image sur le bouton.
- `width` : pour définir la largeur du bouton.
- `height` : pour définir la hauteur du bouton.
- `command` : pour appeler une fonction.
- ...

Python

Les couleurs autorisées

- **Noms de couleurs prédéfinis** : noms de couleurs standards en anglais comme `red`, `green`, `blue`, `black`, `white`, `gray`...
- **Codes hexadécimaux** : comme `#ff0000` pour le rouge...
- **Valeurs RGB**

Python

Pour placer un widget (`Button`) dans le frame

- `place` : place les éléments selon une position calculée par le développeur
- `pack` : place les éléments au centre du frame
- `grid` : place les éléments dans une grille (matrice).

Python

Exemple avec `place`

```
button.place(x=20, y=10, height=20, width=100)
```

Python

Exemple avec `place`

```
button.place(x=20, y=10, height=20, width=100)
```

Exemple avec `grid`

```
root.grid()
button = Button(root, text='Afficher')
button.grid(column=0, row=0)
```

Python

Exemple avec `place`

```
button.place(x=20, y=10, height=20, width=100)
```

Exemple avec `grid`

```
root.grid()
button = Button(root, text='Afficher')
button.grid(column=0, row=0)
```


Exemple avec `pack` qui place les widgets dans le parent en les empilant verticalement ou horizontalement

```
button.pack()
```

Python

Quelques arguments de la méthode `pack()`

- `side` : définit le côté de la fenêtre parent où le widget sera placé. Les valeurs possibles sont `TOP` (par défaut), `BOTTOM`, `LEFT` et `RIGHT`.
- `padx` et `pady` : ajoutent un espace (`padding`) externe horizontal (`padx`) et vertical (`pady`) autour du widget.
- `ipadx` et `ipady` : ajoutent un espace (`padding`) interne horizontal (`ipadx`) et vertical (`ipady`) à l'intérieur du widget, augmentant ainsi sa taille interne.
- ...

Python

Quelques arguments de la méthode `pack()`

- `side` : définit le côté de la fenêtre parent où le widget sera placé. Les valeurs possibles sont `TOP` (par défaut), `BOTTOM`, `LEFT` et `RIGHT`.
- `padx` et `pady` : ajoutent un espace (`padding`) externe horizontal (`padx`) et vertical (`pady`) autour du widget.
- `ipadx` et `ipady` : ajoutent un espace (`padding`) interne horizontal (`ipadx`) et vertical (`ipady`) à l'intérieur du widget, augmentant ainsi sa taille interne.
- ...

Remarques

Les options `padx` et `pady` sont communes aux méthodes `pack()` et `grid()` dans **Tkinter**.

Python

Pour associer une fonction au bouton

```
button = Button(root, text='Afficher', command=afficher_bonjour)
```

Python

Pour associer une fonction au bouton

```
button = Button(root, text='Afficher', command=afficher_bonjour)
```

La fonction `afficher_bonjour` (à placer avant le bouton)

```
def afficher_bonjour():
    print("Bonjour.")
```

Python

Pour désactiver le bouton après le premier clic, on utilise la méthode `config()`

```
def afficher_bonjour():
    print("Bonjour.")
    button.config(state="disabled")
```

Python

Dans une application **Desktop**

- on n'affiche pas les messages dans la console
- on préfère afficher les messages dans les popups

Python

Dans une application **Desktop**

- on n'affiche pas les messages dans la console
- on préfère afficher les messages dans les popups

Remarque

Les popups sont appelés `messagebox` dans **TKinter**.

Python

Commençons par importer `messagebox`

```
from tkinter import Tk, Button, messagebox
```

Python

Commençons par importer `messagebox`

```
from tkinter import Tk, Button, messagebox
```


Attention

`from tkinter import *` ne permet pas d'importer certains **widgets** comme

- `messagebox`
- `ttk`
- ...

Python

Utilisons `messagebox` **pour afficher** `Bonjour`

```
def afficher_bonjour():
    messagebox.showinfo("Message", "Bonjour.")
```

Python

Utilisons `messagebox` pour afficher `Bonjour`

```
def afficher_bonjour():
    messagebox.showinfo("Message", "Bonjour.")
```


Explication

- Le premier paramètre correspond au titre du popup.
- Le deuxième correspond au message

Python

Utilisons `messagebox` pour afficher `Bonjour`

```
def afficher_bonjour():
    messagebox.showinfo("Message", "Bonjour.")
```


Explication

- Le premier paramètre correspond au titre du popup.
- Le deuxième correspond au message

On peut aussi préciser les noms des paramètres

```
messagebox.showinfo(title="Message", message="Bonjour.")
```

Python

Pour modifier l'icône du `messagebox`

```
def afficher_bonjour():
    messagebox.showinfo(title="Message", message="Bonjour.",
       icon=messagebox.WARNING)
```

Python

Pour modifier l'icône du `messagebox`

```
def afficher_bonjour():
    messagebox.showinfo(title="Message", message="Bonjour.",
       icon=messagebox.WARNING)
```


Valeurs possibles pour `icon`

- `messagebox.ERROR`
- `messagebox.INFO` (Icône par défaut pour `showinfo`)
- `messagebox.QUESTION`
- `messagebox.WARNING`

Python

Remarque

`messagebox.showinfo` permet de créer un message d'information avec un seul bouton de confirmation.

Python

Remarque

`messagebox.showinfo` permet de créer un message d'information avec un seul bouton de confirmation.

Autres méthodes de `messagebox`

- `showwarning` = `messagebox.WARNING` + un seul bouton
- `showerror` = `messagebox.ERROR` + un seul bouton
- `askquestion` = `messagebox.QUESTION` + deux boutons (`Oui` et `Non`) [Valeur de retour : `yes` ou `no`]
- `askokcancel` = `messagebox.QUESTION` + deux boutons (`Ok` et `Annuler`) [Valeur de retour : `True` ou `False`]
- `askretrycancel` = `messagebox.WARNING` + deux boutons (`Recommencer` et `Annuler`) [Valeur de retour : `True` ou `False`]
- `askyesno` = `messagebox.QUESTION` + deux boutons (`Oui` et `Non`) [Valeur de retour : `True` ou `False`]
- `askyesnocancel` = `messagebox.QUESTION` + trois boutons (`Oui`, `Non` et `Annuler`) [Valeur de retour : `True`, `False` ou `None`]

Python

Commençons par importer `Label`

```
from tkinter import Tk, Button, messagebox, Label
```

Python

Commençons par importer `Label`

```
from tkinter import Tk, Button, messagebox, Label
```

Pour créer un label

```
label = Label(root, text="Votre nom :")
```

Python

Commençons par importer `Label`

```
from tkinter import Tk, Button, messagebox, Label
```

Pour créer un label

```
label = Label(root, text="Votre nom :")
```


Explication

- Le label a été associé au frame
- Le texte `Votre nom :` sera marqué sur le bouton

Python

Commençons par importer `Entry`

```
from tkinter import Tk, Button, messagebox, Label, Entry
```

Python

Commençons par importer `Entry`

```
from tkinter import Tk, Button, messagebox, Label, Entry
```

Pour créer un label

```
input = Entry(root)
```

Python

Utilisons `grid()` pour placer les éléments

```
root.grid()

button = Button(root, text='Afficher', command=afficher_bonjour)
button.grid(column=1,row=1)

label = Label(root, text="Votre nom :")
label.grid(column=0,row=0)

entry = Entry(root)
entry.grid(column=1, row=0)
```

Python

Pour récupérer la valeur saisie, on utilise `get()`

```
def afficher_bonjour():
    nom = entry.get()
    messagebox.askretrycancel(title="Message", message="Bonjour " + nom)
```

Python

Pour récupérer la valeur saisie, on utilise `get()`

```
def afficher_bonjour():
    nom = entry.get()
    messagebox.askretrycancel(title="Message", message="Bonjour " + nom)
```

Pour placer le curseur dans l'`Entry`

```
entry.focus_set()
```

Python

Pour récupérer la valeur saisie, on utilise `get()`

```
def afficher_bonjour():
    nom = entry.get()
    messagebox.askretrycancel(title="Message", message="Bonjour " + nom)
```

Pour placer le curseur dans l'`Entry`

```
entry.focus_set()
```


Ou

```
entry.focus()
```

Python

Explication

- `focus()` : place le focus sur un widget spécifique et retourne le widget qui l'a actuellement.
- `focus_set()` : force le focus sur le widget appelant, même si la fenêtre principale n'est pas active.
- `focus_get()` : retourne le widget qui a actuellement le focus d'entrée dans l'application. (à appeler depuis le frame. Par exemple : `root.focus_get()`)

Python

Listbox

- une liste d'éléments où l'utilisateur peut sélectionner un ou plusieurs éléments de la liste.
- souvent utilisée pour des listes longues
- initialisée avec la méthode `insert`

Python

Commençons par importer `Listbox`

```
from tkinter import Tk, Listbox
```

Python

Commençons par importer `Listbox`

```
from tkinter import Tk, Listbox
```

Pour créer une `Listbox`

```
listbox = Listbox(root)
```

Python

Pour insérer un élément dans la `Listbox` (l'ordre : `Python`, `PHP`, `Java`)

```
listbox.insert(0, "Python")
listbox.insert(1, "Java")
listbox.insert(2, "PHP")
```

Python

Pour insérer un élément dans la `Listbox` (l'ordre : `Python`, `PHP`, `Java`)

```
listbox.insert(0, "Python")
listbox.insert(1, "Java")
listbox.insert(2, "PHP")
```

Pour placer l'élément à la fin, on peut utiliser la constante `END` (à importer)

```
listbox.insert(END, "C#")
```

Python

Pour insérer un élément dans la `Listbox` (l'ordre : `Python`, `PHP`, `Java`)

```
listbox.insert(0, "Python")
listbox.insert(1, "Java")
listbox.insert(2, "PHP")
```

Pour placer l'élément à la fin, on peut utiliser la constante `END` (à importer)

```
listbox.insert(END, "C#")
```

Pour afficher

```
listbox.pack()
```

Python

Ajoutons un bouton pour la suppression des éléments sélectionnés d'une Listbox

```
bouton_supprimer = Button(root, command=supprimer, text="Supprimer")
bouton_supprimer.pack()
```

Python

Ajoutons un bouton pour la suppression des éléments sélectionnés d'une Listbox

```
bouton_supprimer = Button(root, command=supprimer, text="Supprimer")
bouton_supprimer.pack()
```

Code de la fonction `supprimer` : `ANCHOR` à importer

```
def supprimer():
    listbox.delete(ANCHOR)
```

Python

Ajoutons un bouton pour la suppression de tous les éléments d'une Listbox

```
bouton_supprimer2 = Button(root, command=supprimer_tout, text="
   Supprimer tout")
bouton_supprimer2.pack()
```

Python

Ajoutons un bouton pour la suppression de tous les éléments d'une Listbox

```
bouton_supprimer2 = Button(root, command=supprimer_tout, text="
   Supprimer tout")
bouton_supprimer2.pack()
```

Code de la fonction `supprimer` : `ANCHOR` à importer

```
def supprimer_tout():
    listbox.delete(0, END)
```

Python

Ajoutons un bouton pour l'affichage de l'élément sélectionné d'une Listbox

```
bouton_selectionner = Button(root, command=selectionner, text="Sé
   lectionner")
bouton_selectionner.pack()
```

Python

Ajoutons un bouton pour l'affichage de l'élément sélectionné d'une Listbox

```
bouton_selectionner = Button(root, command=selectionner, text="Sé
   lectionner")
bouton_selectionner.pack()
```

Code de la fonction `supprimer` : `ANCHOR` à importer

```
def selectionner():
    message = "Vous préférez " + listbox.get(ANCHOR)
    messagebox.showinfo("Message", message)
```

Python

Pour autoriser la sélection multiple

```
listbox = Listbox(root, selectmode='multiple')
```

Python

Pour autoriser la sélection multiple

```
listbox = Listbox(root, selectmode='multiple')
```

Ou utiliser la valeur `extended` qui permet de sélectionner une plage d'éléments en utilisant la touche `Shift`, ou de sélectionner des éléments individuels en maintenant la touche `Ctrl` (ou `Command` sur macOS).

```
listbox = Listbox(root, selectmode='extended')
```

Python

Modifions les méthodes de sélection et de suppression pour les adapter à la sélection multiple

```
def supprimer():
    selection = listbox.curselection()
    for i in reversed(selection):
        listbox.delete(i)

def selectionner():
    message = "Vous préférez "
    selection = listbox.curselection()
    for i in selection:
        message += listbox.get(i) + " "
    messagebox.showinfo("Message", message)
```

Python

Spinbox

- une zone de saisie + deux flèches de défilement
- permettant de sélectionner une valeur dans une plage prédéfinie
- pouvant prendre comme entrée un tableau de valeurs

Python

Spinbox

- une zone de saisie + deux flèches de défilement
- permettant de sélectionner une valeur dans une plage prédéfinie
- pouvant prendre comme entrée un tableau de valeurs

Commençons par importer `Spinbox`

```
from tkinter import Tk, Spinbox
```

Python

Pour créer une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10)
spinbox.pack()
```

Python

Pour créer une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10)
spinbox.pack()
```

Explication

- `from_` : permet de spécifier la valeur initiale
- `from_` $\neq$ `from` (utilisé pour importer les modules)
- `to` : permet de spécifier la valeur finale

Python

Pour interdire la saisie de valeur dans une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, state="readonly")
```

Python

Pour interdire la saisie de valeur dans une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, state="readonly")
```

Pour modifier le pas (l'incrémenter de 2 par exemple)

```
spinbox = Spinbox(root, from_=1, to=10, command=afficher, increment=2)
```

Python

Pour interdire la saisie de valeur dans une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, state="readonly")
```

Pour modifier le pas (l'incrémenter de 2 par exemple)

```
spinbox = Spinbox(root, from_=1, to=10, command=afficher, increment=2)
```

Pour associer une fonction au clic sur les boutons d'une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, command=afficher)
```

Python

Pour interdire la saisie de valeur dans une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, state="readonly")
```

Pour modifier le pas (l'incrémenter de 2 par exemple)

```
spinbox = Spinbox(root, from_=1, to=10, command=afficher, increment=2)
```

Pour associer une fonction au clic sur les boutons d'une `Spinbox`

```
spinbox = Spinbox(root, from_=1, to=10, command=afficher)
```

Dans `afficher`, affichons la valeur sélectionnée

```
def afficher():
    messagebox.showinfo("Message", spinbox.get())
```

Python

On peut aussi initialiser une `Spinbox` avec les valeurs d'un tableau

```
valeurs_possibles = [1, 2, 3, 5, 8, 13, 21]

spinbox = Spinbox(root, values=valeurs_possibles, state="readonly")
```

Python

On peut aussi initialiser une `Spinbox` avec les valeurs d'un tableau

```
valeurs_possibles = [1, 2, 3, 5, 8, 13, 21]

spinbox = Spinbox(root, values=valeurs_possibles, state="readonly")
```

Ou un tableau de chaînes de caractère

```
valeurs_possibles = ["Java", "Python", 'PHP', 'C++', 'C#']

spinbox = Spinbox(root, values=valeurs_possibles, state="readonly")
```

Python

Pour créer une `Spinbox` et l'initialiser avec une valeur par défaut au chargement (une valeur comprise entre `from_` et `to`)

```
spinbox_var = IntVar(root)
spinbox_var.set(5)

spinbox = Spinbox(root, from_=1, to=10, textvariable=spinbox_var, state="readonly")
```

Python

Pour créer une `Spinbox` et l'initialiser avec une valeur par défaut au chargement (une valeur comprise entre `from_` et `to`)

```
spinbox_var = IntVar(root)
spinbox_var.set(5)

spinbox = Spinbox(root, from_=1, to=10, textvariable=spinbox_var, state="readonly")
```


Remarque

Pour les chaînes de caractère utiliser `StringVar`...

Python

Exercice

- Dans un nouveau fichier, créer un nouveau frame
- Ses dimensions : 500 (largeur) et 300 (hauteur)
- Ajouter deux `Spinbox` : une pour la largeur et une pour la hauteur
- Les dimensions du frame doivent être calculées en fonction des valeurs présentes dans les `Spinbox`

Python

Création de menu : étapes

1. Création de barre du menu
2. Création de la cascade
3. Attribution des éléments à la cascade

Python

Création de menu : étapes

1. Création de barre du menu
2. Création de la cascade
3. Attribution des éléments à la cascade

Commençons par importer `Menu`

```
from tkinter import Tk, Menu
```

Python

Commençons par créer le menu

```
menu_h = Menu(root)
```

Python

Commençons par créer le menu

```
menu_h = Menu(root)
```

Et les sous menus (leur parent est `menu_h`)

```
menu_fichier = Menu(menu_h)
menu_edition = Menu(menu_h)
```

Python

Commençons par créer le menu

```
menu_h = Menu(root)
```

Et les sous menus (leur parent est `menu_h`)

```
menu_fichier = Menu(menu_h)
menu_edition = Menu(menu_h)
```

Définissons les propriétés des sous-menus

```
menu_h.add_cascade(label="Fichier", menu=menu_fichier)
menu_h.add_cascade(label="Edition", menu=menu_edition)
```

Python

Commençons par créer le menu

```
menu_h = Menu(root)
```

Et les sous menus (leur parent est `menu_h`)

```
menu_fichier = Menu(menu_h)
menu_edition = Menu(menu_h)
```


Définissons les propriétés des sous-menus

```
menu_h.add_cascade(label="Fichier", menu=menu_fichier)
menu_h.add_cascade(label="Edition", menu=menu_edition)
```

Et enfin affichons le menu

```
frame.config(menu=menu_h)
```

Python

Pour créer les sous-menus

```
menu_fichier.add_command(label="Nouveau")
menu_fichier.add_command(label="Ouvrir")
menu_fichier.add_command(label="Quitter")

menu_edition.add_command(label="Couper")
menu_edition.add_command(label="Copier")
menu_edition.add_command(label="Coller")
```

Python

Pour créer les sous-menus

```
menu_fichier.add_command(label="Nouveau")
menu_fichier.add_command(label="Ouvrir")
menu_fichier.add_command(label="Quitter")

menu_edition.add_command(label="Couper")
menu_edition.add_command(label="Copier")
menu_edition.add_command(label="Coller")
```


Explication

- Lancer l'application et vérifier la présence de – – – – – –
- Par défaut les menus sont détachables
- Cliquer sur – – – – – – et vérifier que le menu se détache

Python

Pour désactiver le détachement

```
menu_fichier = Menu(menu_h, tearoff=0)
menu_edition = Menu(menu_h, tearoff=0)
```

Python

Pour désactiver le détachement

```
menu_fichier = Menu(menu_h, tearoff=0)
menu_edition = Menu(menu_h, tearoff=0)
```

Pour quitter l'application en cliquant sur `Quitter`

```
menu_fichier.add_command(label="Quitter", command=frame.destroy)
```

Python

Pour désactiver le détachement

```
menu_fichier = Menu(menu_h, tearoff=0)
menu_edition = Menu(menu_h, tearoff=0)
```

Pour quitter l'application en cliquant sur `Quitter`

```
menu_fichier.add_command(label="Quitter", command=frame.destroy)
```

Associons une fonction au bouton `Ouvrir`

```
menu_fichier.add_command(label="Ouvrir", command=ouvrir)
```

Python

Pour désactiver le détachement

```
menu_fichier = Menu(menu_h, tearoff=0)
menu_edition = Menu(menu_h, tearoff=0)
```

Pour quitter l'application en cliquant sur `Quitter`

```
menu_fichier.add_command(label="Quitter", command=frame.destroy)
```

Associons une fonction au bouton `Ouvrir`

```
menu_fichier.add_command(label="Ouvrir", command=ouvrir)
```

Code de la fonction `ouvrir`

```
def ouvrir():
    file = filedialog.askopenfilename()
    print(file)
```

Python

Pour créer un champ texte multiligne

```
from tkinter import Tk, Text

root = Tk()

root.geometry("1000x600")
root.title('First frame')


text = Text(root, width=40, height=10)
text.pack(pady=10)

root.mainloop()
```

Python

Pour ajouter un texte dans ce champ texte multiligne

```
from tkinter import Tk, Text, END

root = Tk()

root.geometry("1000x600")
root.title('First frame')

chaine = """En Python,
Ceci est un texte multiligne.
"""

text = Text(root, width=40, height=10)
text.insert(END, chaine)
text.pack(pady=10)

root.mainloop()
```

Python

Pour ajouter un scroll

```
from tkinter import Tk, Text, END
from tkinter import scrolledtext

root = Tk()

root.geometry("1000x600")
root.title('First frame')

chaine = """En Python,
Ceci est un texte multiligne.
"""

text =  scrolledtext.ScrolledText(root, width=40, height=10)
text.insert(END, chaine)
text.pack(pady=10)

root.mainloop()
```

Python

Pour placer le curseur dans le champ de texte multiligne

```
from tkinter import Tk, Text, END
from tkinter import scrolledtext

root = Tk()

root.geometry("1000x600")
root.title('First frame')

chaine = """En Python,
Ceci est un texte multiligne.
"""

text =  scrolledtext.ScrolledText(root, width=40, height=10)
text.insert(END, chaine)
text.pack(pady=10)
text.focus()

root.mainloop()
```

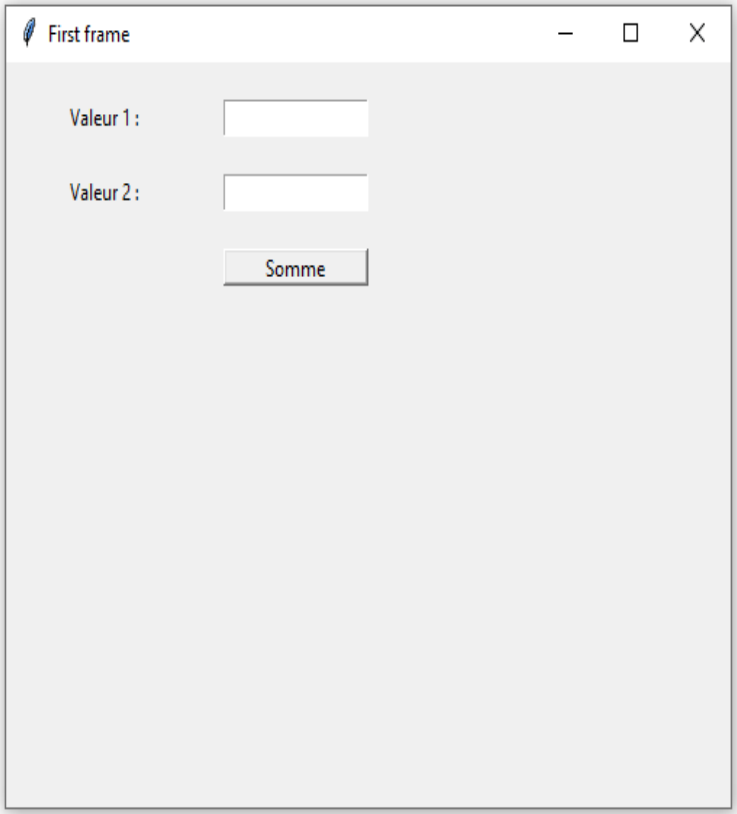
Python

ttk : **Tk t**hemed widgets

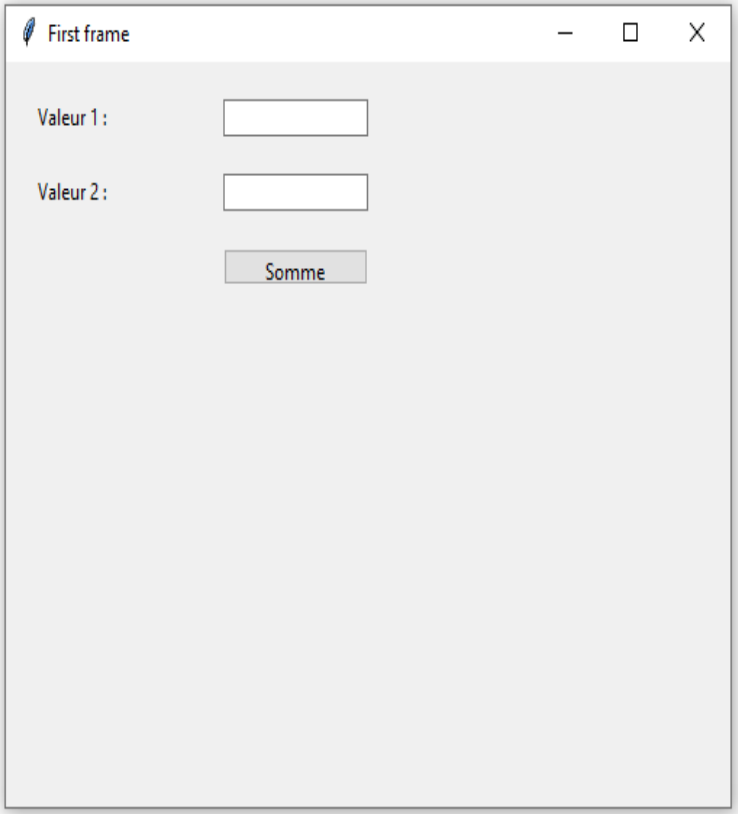
- A été dans la version 2.7 de **Python**
- Améliore le style et offre une apparence plus moderne pour 18 widgets de **tkinter** de base comme
  - **Button**
  - **Entry**
  - **Label**
  - ...
- Inclut également 6 widgets qui n'étaient pas disponibles dans **tkinter** de base comme
  - **Combobox**
  - **Progressbar**
  - ...

Python

Comparaison entre une interface Tkinter et une ttk



Tkinter



ttk

Python

Modifions le code précédent afin d'utiliser `ttk`

```
from tkinter import Tk, messagebox
from tkinter import ttk

root = Tk()

def afficher_bonjour():
    nom = entry.get()
    messagebox.askretrycancel(title="Message", message="Bonjour." + nom)


root.title("First frame")
root.geometry("600x300")
root.grid()

button = ttk.Button(root, text='Afficher', command=afficher_bonjour)
button.grid(column=1, row=2)

label = ttk.Label(root, text="Votre nom :")
label.grid(column=0, row=0)

entry = ttk.Entry(root)
entry.grid(column=1, row=0)

root.mainloop()
```

Python

Combobox

- également connue sous le nom de liste déroulante
- utilisée pour des listes plus courtes et lorsque l'espace à l'écran est limité
- permettant de sélectionner une seule option

Python

Commençons par définir les valeurs à afficher dans la `Combobox`

```
combobox_values = ["Foot", "Tennis", "Basket", "Rugby"]
```

Python

Commençons par définir les valeurs à afficher dans la `Combobox`

```
combobox_values = ["Foot", "Tennis", "Basket", "Rugby"]
```

Créons la `Combobox` avec les valeurs précédentes

```
combobox = ttk.Combobox(root, values=combobox_values)
```

Python

Commençons par définir les valeurs à afficher dans la `Combobox`

```
combobox_values = ["Foot", "Tennis", "Basket", "Rugby"]
```

Créons la `Combobox` avec les valeurs précédentes

```
combobox = ttk.Combobox(root, values=combobox_values)
```

Indiquons l'emplacement de la `Combobox` dans la grille

```
combobox.grid(column=1, row=1)
```

Python

Commençons par définir les valeurs à afficher dans la Combobox

```
combobox_values = ["Foot", "Tennis", "Basket", "Rugby"]
```

Créons la Combobox avec les valeurs précédentes

```
combobox = ttk.Combobox(root, values=combobox_values)
```

Indiquons l'emplacement de la Combobox dans la grille

```
combobox.grid(column=1, row=1)
```

Spécifions l'élément à afficher en premier dans la Combobox

```
combobox.current(0)
```

Python

Pour récupérer la valeur sélectionnée de la combobox

```
def afficher_bonjour():
    nom = entry.get()
    sport = combobox.get()
    texte = nom + " pratique " + sport
    messagebox.askretrycancel(title="Message", message=texte)
```

Python

Pour modifier le style d'un widget ttk, on commence par créer un on objet `Style`

```
style = ttk.Style()
```

Python

Pour modifier le style d'un widget ttk, on commence par créer un on objet `Style`

```
style = ttk.Style()
```

Ensuite, on le configure (l'identifiant doit se terminer par `.TFrame`)

```
style.configure("combo.TFrame", foreground="blue", background="red")
```

Python

Pour modifier le style d'un widget ttk, on commence par créer un on objet `Style`

```
style = ttk.Style()
```

Ensuite, on le configure (l'identifiant doit se terminer par `.TFrame`)

```
style.configure("combo.TFrame", foreground="blue", background="red")
```


Et enfin on l'associe au widget

```
combobox = ttk.Combobox(root, style="combo.TFrame", values=
   combobox_values)
```

Python

Treeview

- conçu pour représenter des structures complexes
- permettant d'afficher des données de manière tabulaire

Python

Créons la `Treeview`

```
treeview = ttk.Treeview(root, columns=('ID', 'Nom', 'Age'), show='
   headings')
```

Python

Créons la `Treeview`

```
treeview = ttk.Treeview(root, columns=('ID', 'Nom', 'Age'), show='
   headings')
```


Explication

- `columns=('ID', 'Nom', 'Age')` : permet de lister les colonnes
- `show='headings'` : afficher uniquement les en-têtes des colonnes, et non la colonne de l'arbre par défaut qui est utilisée pour afficher la structure hiérarchique (les "branches" de l'arbre)

Python

Remplaçons le nom de chaque colonne par la valeur que l'on souhaite afficher et sa largeur

```
treeview.heading('ID', text='ID')
treeview.column('ID', width=100)
treeview.heading('Nom', text='Nom')
treeview.column('Nom', width=100)
treeview.heading('Age', text='Age')
treeview.column('Age', width=100)
```

Python

Remplaçons le nom de chaque colonne par la valeur que l'on souhaite afficher et sa largeur

```
treeview.heading('ID', text='ID')
treeview.column('ID', width=100)
treeview.heading('Nom', text='Nom')
treeview.column('Nom', width=100)
treeview.heading('Age', text='Age')
treeview.column('Age', width=100)
```

Remplissons `Treeview` avec quelques valeurs de test

```
donnees = [
    (1, "Alice", 30),
    (2, "Mary", 25),
    (3, "John", 35)
]
for id, nom, age in donnees:
    treeview.insert('', END, values=(id, nom, age))

treeview.pack()
```

Python

Pour récupérer les éléments sélectionnés, ajoutons le bouton suivant

```
btn_afficher = Button(root, text="Afficher", command=afficher_selection
   )
btn_afficher.pack()
```

Python

Pour récupérer les éléments sélectionnés, ajoutons le bouton suivant

```
btn_afficher = Button(root, text="Afficher", command=afficher_selection
   )
btn_afficher.pack()
```

Récupérons puis affichons la sélection

```
def afficher_selection():
    ids = treeview.selection()
    if ids:
        for id in ids:
            item = treeview.item(id)
            print(item['values'])
    else:
        print('Aucun élément sélectionné')
```

Python

Pour supprimer les éléments sélectionnés, ajoutons le bouton suivant

```
btn_supprimer = Button(root, text="Supprimer", command=
   supprimer_selection)
btn_supprimer.pack()
```

Python

Pour supprimer les éléments sélectionnés, ajoutons le bouton suivant

```
btn_supprimer = Button(root, text="Supprimer", command=
   supprimer_selection)
btn_supprimer.pack()
```

Récupérons puis affichons la sélection

```
def supprimer_selection():
    selections = treeview.selection()
    for selection in selections:
        treeview.delete(selection)
    print("éléments sélectionnés supprimés.")
```

Python

Commençons par interdire la sélection multiple

```
treeview = ttk.Treeview(root, columns=('ID', 'Nom', 'Age'), show='headings', selectmode='browse
   ')
```

Python

Commençons par interdire la sélection multiple

```
treeview = ttk.Treeview(root, columns=('ID', 'Nom', 'Age'), show='headings', selectmode='browse
   ')
```

Pour modifier les éléments sélectionnés, ajoutons le bouton suivant

```
btn_modifier = Button(root, text="Modifier", command=modifier_selection)
btn_modifier.pack()
```

Python

Commençons par interdire la sélection multiple

```
treeview = ttk.Treeview(root, columns=('ID', 'Nom', 'Age'), show='headings', selectmode='browse
   ')
```

Pour modifier les éléments sélectionnés, ajoutons le bouton suivant

```
btn_modifier = Button(root, text="Modifier", command=modifier_selection)
btn_modifier.pack()
```


Récupérons puis affichons la sélection

```
def modifier_selection():
    selection = treeview.selection()
    if selection:
        premier_element = selection[0]
        nouvelle_valeur = (4, "Dalton", "Jack")
        treeview.item(premier_element, values=nouvelle_valeur)
        print("élément sélectionné modifié.")
    else:
        print("Aucun élément sélectionné.")
```

Python

Créons la `Progressbar` de la manière suivante

```
progressbar = ttk.Progressbar(root, orient="horizontal", length=100)
progressbar.pack()
```

Python

Créons la `Progressbar` de la manière suivante

```
progressbar = ttk.Progressbar(root, orient="horizontal", length=100)
progressbar.pack()
```

Explication

- `orient` : `horizontal` ou `vertical`
- `length` : pour définir la longueur en pixel

Python

Pour gérer la progression, ajoutons le bouton suivant

```
btn_plus = Button(root, text="Plus", command=plus)
btn_plus.pack()
```

Python

Pour gérer la progression, ajoutons le bouton suivant

```
btn_plus = Button(root, text="Plus", command=plus)
btn_plus.pack()
```

La méthode `step` permet d'incrémenter la valeur de la barre

```
def plus():
    progressbar.step(10)
```

Python

Pour gérer la progression, ajoutons le bouton suivant

```
btn_plus = Button(root, text="Plus", command=plus)
btn_plus.pack()
```

La méthode `step` permet d'incrémenter la valeur de la barre

```
def plus():
    progressbar.step(10)
```


La même chose peut être faite ainsi

```
def plus():
    progressbar['value'] += 10
```

Python

Pour répéter l'appel de la fonction `step`, on peut utiliser la fonction `start`

```
def plus():
    progressbar.start(1000)
```

Python

Pour répéter l'appel de la fonction `step`, on peut utiliser la fonction `start`

```
def plus():
    progressbar.start(1000)
```


Explication

- Le paramètre passé à `start` est l'intervalle en milli-secondes
- `start` appelle la fonction `step(1)` avec la valeur 1

Python

Pour arrêter la progression, ajoutons le bouton suivant

```
btn_stop = Button(root, text="Stop", command=stop)
btn_stop.pack()
```

Python

Pour arrêter la progression, ajoutons le bouton suivant

```
btn_stop = Button(root, text="Stop", command=stop)
btn_stop.pack()
```

La méthode `step` permet d'incrémenter la valeur de la barre

```
def stop():
    progressbar.stop()
```

Python

Modifions le mode de la `Progressbar` pour qu'elle s'anime en continu

```
progressbar = ttk.Progressbar(root, orient="vertical", length=100, mode
   ="indeterminate")
```

Python

Modifions le mode de la `Progressbar` pour qu'elle s'anime en continu

```
progressbar = ttk.Progressbar(root, orient="vertical", length=100, mode
   ="indeterminate")
```

Autres propriétés de la `Progressbar`

- `value`
- `maximum`

Python

Considérons le frame suivant

```
from tkinter import Tk

width = 600
height = 300


root = Tk()
root.geometry(f"{width}x{height}")
root.title('Destroy')

root.mainloop()
```

Python

Objectif

- Ajouter un premier bouton pour supprimer le frame
- Ajouter un deuxième bouton pour supprimer un widget

Python

Ajoutons un premier bouton qui permet de fermer le frame

```
from tkinter import Tk, Button

width = 600
height = 300


root = Tk()
root.geometry(f"{width}x{height}")
root.title('Destroy')

btn1 = Button(root, text="Button 1", command=root.destroy)
btn1.pack(pady=10)

root.mainloop()
```

Python

Il est aussi possible d'utiliser `close`

```
from tkinter import Tk, Button

width = 600
height = 300


root = Tk()
root.geometry(f"{width}x{height}")
root.title('Destroy')

btn1 = Button(root, text="Button 1", command=root.close)
btn1.pack(pady=10)

root.mainloop()
```

Python

`close()` vs `destroy()`

- `close` : utilisée pour fermer un frame mais ne détruit pas complètement l'objet. Le frame peut être rouvert ou réutilisé ultérieurement sans avoir à recréer complètement l'objet.
- `destroy` : utilisée pour détruire complètement l'objet. Toutes les ressources associées à ce frame sont libérées de la mémoire et le frame ne peut plus être réutilisé ou rouvert.

Python

Ajoutons un deuxième bouton qui permet de supprimer un widget (un bouton)

```
from tkinter import Tk, Button

width = 600
height = 300


root = Tk()
root.geometry(f"{width}x{height}")
root.title('Destroy')

btn1 = Button(root, text="Button 1", command=root.destroy)
btn1.pack(pady=10)

btn2 = Button(root, text="Button 2", command=btn1.destroy)
btn2.pack(pady=10)

root.mainloop()
```

Python

Pour supprimer un widget, on peut aussi utiliser la méthode `pack_forget()`

```
from tkinter import Tk, Button

width = 600
height = 300


def cacher():
    btn1.pack_forget()


root = Tk()
root.geometry(f"{width}x{height}")
root.title('Destroy')

btn1 = Button(root, text="Button 1", command=root.destroy, bg='red')
btn1.pack(pady=10)

btn2 = Button(root, text="Button 2", command=cacher)
btn2.pack(pady=10)

root.mainloop()
```

Python

`destroy()` vs `pack_forget()`

- `destroy()` : utilisée pour détruire complètement un widget et libérer les ressources associées. Le widget n'existe plus du tout dans l'application.
- `pack_forget()` : utilisée pour retirer temporairement un widget de la fenêtre sans le détruire complètement. Le widget peut être réaffiché en utilisant `pack()`, `grid()` ou `place()`.

Python

`destroy()` vs `pack_forget()`

- `destroy()` : utilisée pour détruire complètement un widget et libérer les ressources associées. Le widget n'existe plus du tout dans l'application.
- `pack_forget()` : utilisée pour retirer temporairement un widget de la fenêtre sans le détruire complètement. Le widget peut être réaffiché en utilisant `pack()`, `grid()` ou `place()`.

Remarques

- `place_forget()` et `grid_forget()` sont similaires à `pack_forget()` mais spécifiques aux méthodes `place()` et `grid()` respectivement.
- Elles sont donc utilisées pour retirer temporairement un widget de la fenêtre sans le détruire complètement, de la même manière que `pack_forget()`.

Python

Évènements

- certains sont associés à des widgets via l'attribut `command`
- d'autres sont associés via la méthode `bind()` qui prend deux paramètres :
  - le nom de l'événement
  - la callback (gestionnaire d'événements) qui
    - prend un argument (un objet événement)
    - définit ce qui doit se passer lorsque l'événement est déclenché

Python

Considérons le frame suivant

```
from tkinter import Tk, Label

root = Tk()
root.title("First frame")
root.geometry("500x300")

label = Label(root, text="Texte qui change de couleur")
label.pack(pady=20)

root.mainloop()
```

Python

Considérons le frame suivant

```
from tkinter import Tk, Label

root = Tk()
root.title("First frame")
root.geometry("500x300")

label = Label(root, text="Texte qui change de couleur")
label.pack(pady=20)

root.mainloop()
```


Objectif

Attribuer une couleur différente chaque fois qu'on clique sur un des trois boutons de la souris.

Python

Commençons par associer une fonction à chaque bouton de souris

```
label.bind("<Button-1>", afficher_rouge)
label.bind("<Button-2>", afficher_bleu)
label.bind("<Button-3>", afficher_vert)
```

Python

Commençons par associer une fonction à chaque bouton de souris

```
label.bind("<Button-1>", afficher_rouge)
label.bind("<Button-2>", afficher_bleu)
label.bind("<Button-3>", afficher_vert)
```

Objectif

- `<Button-1>` : bouton gauche
- `<Button-2>` : bouton milieu (ou roulette)
- `<Button-3>` : bouton droite

Python

Définissons les fonctions associées aux 3 boutons

```
def afficher_rouge(event):
    label.config(bg="red")
    print(event)


def afficher_bleu(event):
    label.config(bg="blue")


def afficher_vert(event):
    label.config(bg="green")
```

Python

Nous pouvons également associer une seule fonction à tous les boutons

```
label.bind("<Button-1>", afficher_rouge)
label.bind("<Button-2>", afficher_bleu)
label.bind("<Button-3>", afficher_vert)
```

Python

Nous pouvons également associer une seule fonction à tous les boutons

```
label.bind("<Button-1>", afficher_rouge)
label.bind("<Button-2>", afficher_bleu)
label.bind("<Button-3>", afficher_vert)
```

Définissons les fonctions associées aux 3 boutons

```
def changer_couleur(event):
    if event.num == 1:
        couleur = "red"
    elif event.num == 2:
        couleur = "blue"
    elif event.num == 3:
        couleur = "green"
    label.config(bg=couleur)
```

Python

Le code précédent peut être simplifié en utilisant les `lambda`

```
label.bind("<Button-1>", lambda event: changer_couleur("red", event))
label.bind("<Button-2>", lambda event: changer_couleur("blue", event))
label.bind("<Button-3>", lambda event: changer_couleur("green", event))
```

Python

Le code précédent peut être simplifié en utilisant les `lambda`

```
label.bind("<Button-1>", lambda event: changer_couleur("red", event))
label.bind("<Button-2>", lambda event: changer_couleur("blue", event))
label.bind("<Button-3>", lambda event: changer_couleur("green", event))
```

La fonction `changer_couleur` devient

```
def changer_couleur(couleur, event):
    label.config(bg=couleur)
```

Python

Le code précédent peut être simplifié en utilisant les `lambda`

```
label.bind("<Button-1>", lambda event: changer_couleur("red", event))
label.bind("<Button-2>", lambda event: changer_couleur("blue", event))
label.bind("<Button-3>", lambda event: changer_couleur("green", event))
```

La fonction `changer_couleur` devient

```
def changer_couleur(couleur, event):
    label.config(bg=couleur)
```


Remarque

La callback liée à un événement avec `bind` ne permet pas de passer des arguments supplémentaires directement.

Python

Autres évènements

- **Mouvement de la souris** : `<Motion>` pour le mouvement de la souris sur un widget, `<Enter>` et `<Leave>` pour la souris entrant ou quittant la zone du widget.
- **Pressions de touches** : `<KeyPress>`, `<KeyRelease>` ou des événements spécifiques comme `<KeyPress-A>` pour la touche A.
- **Événements de focus** : `<FocusIn>`, `<FocusOut>` pour le focus gagné ou perdu.
- **Événements de widgets spécifiques** : Certains widgets ont des événements spécifiques, comme `<ListboxSelect>` pour une sélection dans une `Listbox`.
- **Événements pour des combinaisons personnalisées** : par exemple `<Control-Shift-D>` pour CTRL + Shift + D...
- ...

Python

Considérons le frame suivant

```
import tkinter as tk

width = 600
height = 300

root = tk.Tk()
root.geometry(f"{width}x{height}")
root.title('Validate a frame')

label = tk.Label(root, text="Entrez un nombre :")
label.pack()

reg = root.register(checks_number)

entry = tk.Entry(root)
entry.pack()

root.mainloop()
```

Python

Validation des données

- N'autoriser que les valeurs numériques
- Annuler les autres saisies

Python

Déclarons la fonction de validation suivante qui retourne `True` si la valeur saisie est valide, `False` sinon

```
def checks_number(input):
    print(input)
    if input.isdigit():
        return True
    elif input is "":
        return True
    else:
        return False
```

Python

Enregistrons cette fonction de validation

```
reg = root.register(checks_number)
```

Python

Enregistrons cette fonction de validation

```
reg = root.register(checks_number)
```

Associons la fonction de validation à notre champ de saisie

```
entry = tk.Entry(root, validate="key", validatecommand=(reg, '%P'))
```

Python

Enregistrons cette fonction de validation

```
reg = root.register(checks_number)
```

Associons la fonction de validation à notre champ de saisie

```
entry = tk.Entry(root, validate="key", validatecommand=(reg, '%P'))
```


Explication

- `validate="key"` : signifie que la fonction de validation sera appelée chaque fois qu'une touche est pressée dans le champ de saisie.
- `%P` : code de substitution utilisé par **tkinter** pour passer le contenu actuel du champ de saisie (après que la touche a été pressé) à la fonction de validation.

Python

Autres valeurs autorisées pour la propriété `validate`

- `focus` : permet de déclencher le validateur quand le champ récupère ou perd le focus
- `focusin` : permet de déclencher le validateur quand le champ récupère le focus
- `focusout` : permet de déclencher le validateur quand le champ perd le focus
- `key` : permet de déclencher le validateur chaque fois qu'une touche est pressée
- `all` : permet de déclencher le validateur pour tous les évènements précédents
- `none` : permet de désactiver les validateurs

Python

Autres codes de substitution

- `%W` : renvoie le nom du champ
- `%V` : renvoie la raison de la callback : `focusin`, `focusout`, `key`...
- `%s` : renvoie le texte avant le changement
- `%P` : renvoie la valeur actuelle du champ de saisie
- `%d` : renvoie 1 si un caractère est inséré, -1 si un caractère est supprimé, et 0 pour toute autre action.
- ...

Python

À l'exécution

- Si nous saisissons des chiffres, la callback retourne `True` et la valeur est autorisée dans le champ de saisie.
- Sinon (si nous saisissons autre chose), la callback retourne `False` et la valeur n'est pas autorisée à être saisie dans le champ de saisie.
- L'insertion et la suppression de chiffres sont également autorisées dans le champ de saisie.

Python

Commençons par préparer la structure générale de notre application

```
from tkinter import *

width = 600
height = 300

root = Tk()
root.geometry(f"{width}x{height}")
root.title('Switching frame')

root.mainloop()
```

Python

Commençons par préparer la structure générale de notre application

```
from tkinter import *

width = 600
height = 300

root = Tk()
root.geometry(f"{width}x{height}")
root.title('Switching frame')

root.mainloop()
```


Objectif

- Avoir deux frames contenant chacun un bouton
- Le clic sur le bouton d'un frame permet d'afficher l'autre frame

Python

Dans ce même fichier, déclarons les deux autres frames

```
# premier frame
f1 = Frame(root, bg='red')
bouton1 = Button(f1, text='Go to frame 2')
bouton1.pack()
lable1 = Label(f1, text='FRAME 1')
lable1.pack()

# deuxième frame
f2 = Frame(root, bg='blue')
label2 = Label(f2, text='FRAME 2')
label2.pack()
bouton1 = Button(f2, text='Go to frame 1')
bouton1.pack()
```

Python

Placer les deux frames dans le frame principal (même dimension pour les trois)

```
from tkinter import *

width=600
height=300

root = Tk()
root.geometry(f"{width}x{height}")
root.title('Switching frame')

f1 = Frame(root, bg='red')
bouton1 = Button(f1, text='Go to frame 2'))
bouton1.pack()
lable1 = Label(f1, text='FRAME 1')
lable1.pack()

f2 = Frame(root, bg='blue')
label2 = Label(f2, text='FRAME 2')
label2.pack()
bouton1 = Button(f2, text='Go to frame 1')
bouton1.pack()

for frame in (f1, f2):
    frame.place(x=0, y=0, width=width, height=height)

root.mainloop()
```

Python

Ajoutons maintenant la fonction qui permet de switcher les frames

```
from tkinter import *

def switch_frame(frame1):
    frame1.tkraise()

width = 600
height = 300

root = Tk()
root.geometry(f"{width}x{height}")
root.title('Switching frame')

f1 = Frame(root, bg='red')
bouton1 = Button(f1, text='Go to frame 2', command=lambda: switch_frame(f2))
bouton1.pack()
lable1 = Label(f1, text='FRAME 1')
lable1.pack()

f2 = Frame(root, bg='blue')
label2 = Label(f2, text='FRAME 2')
label2.pack()
bouton1 = Button(f2, text='Go to frame 1', command=lambda: switch_frame(f1))
bouton1.pack()

for frame in (f1, f2):
    frame.place(x=0, y=0, width=width, height=height)

root.mainloop()
```

Python

Commençons par préparer la structure générale de notre application

```
from tkinter import *
from tkinter.ttk import *

root = Tk()
root.geometry("400x400")
root.title("First frame")


def second_frame():
    pass

label = Label(root, text="First frame")
label.pack(pady=10)

btn = Button(root, text="Second frame", command=second_frame)
btn.pack(pady=10)

mainloop()
```

Python

Commençons par préparer la structure générale de notre application

```
from tkinter import *
from tkinter.ttk import *

root = Tk()
root.geometry("400x400")
root.title("First frame")


def second_frame():
    pass

label = Label(root, text="First frame")
label.pack(pady=10)

btn = Button(root, text="Second frame", command=second_frame)
btn.pack(pady=10)

mainloop()
```

Objectif

Afficher un deuxième frame tout en gardant le premier ouvert

Python

`TopLevel` **permet de créer un deuxième frame (avec ses propres titre et dimension) tout en gardant le premier ouvert**

```
from tkinter import *
from tkinter.ttk import *

root = Tk()
root.geometry("400x400")
root.title("First frame")


def second_frame():
    frame = Toplevel(root)
    frame.title("Second frame")
    frame.geometry("200x200")
    Label(frame, text="This is a new window").pack()


label = Label(root, text="First frame")
label.pack(pady=10)

btn = Button(root, text="Second frame", command=second_frame)
btn.pack(pady=10)

mainloop()
```

Python

Ajoutons deux boutons dans le premier frame pour cacher et afficher le frame

```
btn_h = Button(root, text="Hide frame", command=hide_frame)
btn_h.pack(pady=10)

btn_s = Button(root, text="Show frame", command=show_frame)
btn_s.pack(pady=10)
```

Python

Ajoutons deux boutons dans le premier frame pour cacher et afficher le frame

```
btn_h = Button(root, text="Hide frame", command=hide_frame)
btn_h.pack(pady=10)

btn_s = Button(root, text="Show frame", command=show_frame)
btn_s.pack(pady=10)
```

Dans la fonction `second_frame()`, déclarons `frame` comme étant global

```
def second_frame():
    global frame
    frame = Toplevel(root)
    frame.title("Second frame")
    frame.geometry("200x200")
    Label(frame, text="This is a new window").pack()
```

Python

Définissons maintenant les fonctions qui permettrons de cacher ou afficher le deuxième frame

```
def hide_frame():
    frame.withdraw()


def show_frame():
    frame.deiconify()
```

Python

Considérons le frame suivant

```
from tkinter import Tk

root = Tk()

root.geometry("1000x600")
```

Python

Considérons le frame suivant

```
from tkinter import Tk

root = Tk()

root.geometry("1000x600")
```


Pour afficher une page Web dans un frame Tkinter nous devrions installer

```
pip install pywebview
```

Python

Utilisons `pywebview` pour visualiser le contenu d'une application Web

```
from tkinter import Tk
import webview as web

root = Tk()

root.geometry("1000x600")

web.create_window('Achref EL MOUELHI', 'http://elmouelhia.free.fr/')
web.start()
```

Python

Pour créer un lien qui permet d'ouvrir une page web dans le navigateur

```
from tkinter import Tk, Button
import webbrowser as browser


def open_browser():
    browser.open("http://elmouelhia.free.fr")


root = Tk()

root.geometry("1000x600")
root.title('Lien vers une page web')

button = Button(root, text='Ouvrir ma page', command=open_browser)
button.pack()

root.mainloop()
```

Python

Dans une application **Tkinter**, on a

- un Frame Principal (ou Root Window)
  - Créé généralement en instanciant `Tk()`
  - Servant de conteneur principal pour tous les autres widgets et frames
  - Gérant des aspects globaux comme la barre de titre, la taille et la position de la fenêtre...
- plusieurs Secondary Frame (ou Child Frame)
  - Créés à l'intérieur du frame principal ou d'autres frames.
  - Instanciés en utilisant `Frame(master, **options)` : `master` est le frame parent (qui peut principal ou secondaire)
  - Héritant de certaines propriétés de leur frame parent, mais peuvent aussi avoir des caractéristiques propres.

Python

Comment ça marche ?

- Un frame $\Rightarrow$ un fichier
- Deux solutions possibles
  - Un frame $\Rightarrow$ une fonction
  - Un frame $\Rightarrow$ une classe (la solution la plus courante)

Python

Pour l'exemple

Nous allons transformer les derniers de switching en plusieurs fichiers/frames

- `frame1.py`
- `frame2.py`
- `root.py`
- `main.py`

Python

Commençons par `root.py`

```
from tkinter import Tk

def create_root(width, height):
    root = Tk()
    root.geometry(f"{width}x{height}")
    root.title('Switching frame')
    return root
```

Python

Ensuite `frame1.py`

```
from tkinter import Frame, Button, Label

def create_frame1(root, switch_frame):
    f1 = Frame(root, bg='red')
    bouton1 = Button(f1, text='Go to frame 2', command=lambda: switch_frame('f2'))
    bouton1.pack()
    label1 = Label(f1, text='FRAME 1')
    label1.pack()
    return f1
```

Python

Ensuite `frame1.py`

```
from tkinter import Frame, Button, Label

def create_frame1(root, switch_frame):
    f1 = Frame(root, bg='red')
    bouton1 = Button(f1, text='Go to frame 2', command=lambda: switch_frame('f2'))
    bouton1.pack()
    label1 = Label(f1, text='FRAME 1')
    label1.pack()
    return f1
```

Et `frame2.py`

```
from tkinter import Frame, Button, Label

def create_frame2(root, switch_frame):
    f2 = Frame(root, bg='blue')
    bouton2 = Button(f2, text='Go to frame 1', command=lambda: switch_frame('f1'))
    bouton2.pack()
    label2 = Label(f2, text='FRAME 2')
    label2.pack()
    return f2
```

Python

Et enfin `main.py`

```
from tkinter import *
from root import create_root
from frame1 import create_frame1
from frame2 import create_frame2

width = 600
height = 300


def switch_frame(frame_name):
    frame = frames[frame_name]
    frame.tkraise()


root = create_root(width, height)

frames = {}

f1 = create_frame1(root, switch_frame)
f2 = create_frame2(root, switch_frame)

frames['f1'] = f1
frames['f2'] = f2

for frame in frames.values():
    frame.place(x=0, y=0, width=width, height=height)

root.mainloop()
```

Python

Pour l'exemple

- Nous allons transformer les frames suivants en classes
  - `frame_spinbox.py`
  - `frame_progressbar.py`
  - `frame_menu.py`
- Un frame principal sera créé

Python

Commençons par créer une classe `FrameSpinbox` qui hérite de `Frame`

```
from tkinter import Frame


class FrameSpinbox(Frame):
    def __init__(self, master=None):
        super().__init__(master)
```

Python

Dans le constructeur, déclarons les widgets comme propriétés

```
from tkinter import  Spinbox, IntVar, Frame


class FrameSpinbox(Frame):
    def __init__(self, master=None):
        super().__init__(master)

        spinbox_var = IntVar(self)
        spinbox_var.set(5)

        self.spinbox = Spinbox(self, from_=1, to=10, textvariable=
          spinbox_var, state="readonly", command=self.afficher)
        self.spinbox.pack()
```

Python

Les fonctions associées aux évènements seront déclarées comme méthodes

```
from tkinter import Spinbox,  messagebox, IntVar, Frame


class FrameSpinbox(Frame):
    def __init__(self, master=None):
        super().__init__(master)

        spinbox_var = IntVar(self)
        spinbox_var.set(5)

        self.spinbox = Spinbox(self, from_=1, to=10,  textvariable=
           spinbox_var, state="readonly", command=self.afficher)
        self.spinbox.pack()

    def afficher(self):
        messagebox.showinfo("Message", self.spinbox.get())
```

Python

De la même manière, on transforme `frame_progressbar.py`

```
from tkinter import Button,  Frame
from tkinter import ttk


class FrameProgressbar(Frame):
    def __init__(self, master=None):
        super().__init__(master)

        self.progressbar = ttk.Progressbar(self, orient="vertical", length=100)
        self.progressbar.pack()
        btn_plus = Button(self, text="Plus", command=self.plus)
        btn_plus.pack()

        btn_stop = Button(self, text="Stop", command=self.stop)
        btn_stop.pack()

    def plus(self):
        self.progressbar.step(10)

    def stop(self):
        self.progressbar.stop()
```

De la même manière, on transforme `frame_menu.py` : on ajoute une cascade pour les exemples

```
from tkinter import Menu, filedialog
from frame_spinbox import FrameSpinbox
from frame_progressbar import FrameProgressbar

class MenuBar(Menu):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master

        menu_fichier = Menu(self, tearoff=0)
        menu_edition = Menu(self, tearoff=0)
        menu_exemple = Menu(self, tearoff=0)

        self.add_cascade(label="Fichier", menu=menu_fichier)
        self.add_cascade(label="Edition", menu=menu_edition)
        self.add_cascade(label="Exemples", menu=menu_exemple)

        menu_exemple.add_command(label="Spinbox", command=lambda: master.show_frame(
          FrameSpinbox))
        menu_exemple.add_command(label="Progressbar", command=lambda: master.show_frame(
          FrameProgressbar))

        menu_fichier.add_command(label="Nouveau")
        menu_fichier.add_command(label="Ouvrir", command=self.ouvrir)
        menu_fichier.add_command(label="Quitter", command=exit)

        menu_edition.add_command(label="Couper")
        menu_edition.add_command(label="Copier")
        menu_edition.add_command(label="Coller")

    def ouvrir(self):
        file = filedialog.askopenfilename()
```

Python

Dans `main_app`, créons le frame principal qui hérite de `Tk`

```
from tkinter import Tk

class MainApp(Tk):
    def __init__(self):
        super().__init__()
```

Python

Spécifions les propriétés du frame principal

```
from tkinter import Tk

class MainApp(Tk):
    def __init__(self):
        super().__init__()

        self.title("Application Tkinter")
        self.geometry("600x300")
        self.current_frame = None
```

Python

Configurons le menu

```
from tkinter import Tk
from frame_spinbox import FrameSpinbox
from frame_menu import MenuBar

class MainApp(Tk):
    def __init__(self):
        super().__init__()

        self.title("Application Tkinter")
        self.geometry("600x300")
        self.current_frame = None

        self.menu_bar = MenuBar(self)
        self.config(menu=self.menu_bar)
```

Python

Affichons `FrameSpinbox` comme Frame par défaut

```
from tkinter import Tk
from frame_spinbox import FrameSpinbox
from frame_menu import MenuBar

class MainApp(Tk):
    def __init__(self):
        super().__init__()

        self.title("Application Tkinter")
        self.geometry("600x300")
        self.current_frame = None

        # Configurer le menu
        self.menu_bar = MenuBar(self)
        self.config(menu=self.menu_bar)

        self.show_frame(FrameSpinbox)
```

Python

Définissons la fonction `show_frame` la responsable d'afficher les frames secondaires dans le frame principal

```
from tkinter import Tk
from frame_spinbox import FrameSpinbox
from frame_menu import MenuBar

class MainApp(Tk):
    def __init__(self):
        super().__init__()

        self.title("Application Tkinter")
        self.geometry("600x300")
        self.current_frame = None

        # Configurer le menu
        self.menu_bar = MenuBar(self)
        self.config(menu=self.menu_bar)

        # Afficher FrameSpinbox comme Frame par défaut
        self.show_frame(FrameSpinbox)

    def show_frame(self, frame_class):
        if self.current_frame is not None:
            self.current_frame.destroy()
        self.current_frame = frame_class(self)
        self.current_frame.pack()
```

Python

Et enfin le code permettant de lancer l'application

```
from tkinter import Tk
from frame_spinbox import FrameSpinbox
from frame_menu import MenuBar

class MainApp(Tk):
    def __init__(self):
        super().__init__()

        self.title("Application Tkinter")
        self.geometry("600x300")
        self.current_frame = None

        # Configurer le menu
        self.menu_bar = MenuBar(self)
        self.config(menu=self.menu_bar)

        # Afficher FrameSpinbox comme Frame par défaut
        self.show_frame(FrameSpinbox)

    def show_frame(self, frame_class):
        # Détruire le frame actuel, s'il existe
        if self.current_frame is not None:
            self.current_frame.destroy()
        self.current_frame = frame_class(self)
        self.current_frame.pack()

if __name__ == "__main__":
    app = MainApp()
    app.mainloop()
```