

⌚ Fiche mémo — Tkinter : Widgets essentiels & Internationalisation (i18n)

Cette fiche regroupe les deux aspects fondamentaux pour construire des interfaces Tkinter modernes et multilingues :

les widgets essentiels et **⌚ la gestion des langues (i18n)**.

1) Widgets essentiels de Tkinter

Tkinter fournit de nombreux widgets simples et légers permettant de composer une interface graphique.

💻 Les plus courants

Widget	Description	Exemple d'usage
Label	Affiche un texte ou une image.	<code>tk.Label(fenetre, text="Bonjour")</code>
Button	Bouton cliquable déclenchant une commande.	<code>tk.Button(fenetre, text="OK", command=action)</code>
Entry	Champ de saisie à une ligne.	<code>tk.Entry(fenetre)</code>
Text	Zone de texte multilignes.	<code>tk.Text(fenetre, height=10, width=40)</code>
LabelFrame	Conteneur avec un titre encadrant d'autres widgets.	<code>tk.LabelFrame(fenetre, text="Options")</code>
Frame	Conteneur de base pour structurer la mise en page.	<code>tk.Frame(fenetre)</code>
Checkbutton	Case à cocher (booléenne).	<code>tk.Checkbutton(fenetre, text="Activer", variable=etat)</code>
Radiobutton	Bouton radio (choix unique).	<code>tk.Radiobutton(fenetre, text="Option A", variable=choix, value=1)</code>
OptionsMenu	Menu déroulant simple.	<code>tk.OptionMenu(fenetre, var, *options)</code>
Scrollbar	Barre de défilement verticale ou horizontale.	<code>tk.Scrollbar(zone)</code>

2) Positionnement des widgets

Trois systèmes principaux permettent de disposer les widgets dans une fenêtre :

Méthode	Caractéristiques	Exemple
<code>.pack()</code>	Simple, empile les widgets les uns après les autres.	<code>label.pack(padx=10, pady=5)</code>
<code>.grid()</code>	Disposition en lignes et colonnes (tableau).	<code>bouton.grid(row=0, column=1)</code>

Méthode	Caractéristiques	Exemple
.place()	Position absolue en pixels (rarement recommandé).	widget.place(x=50, y=20)

Bonnes pratiques :

- Ne jamais mélanger pack() et grid() dans le même conteneur.
- Utiliser Frame ou LabelFrame pour structurer visuellement les zones.
- Utiliser expand=True et fill='both' pour un redimensionnement fluide.

3) Gestion dynamique de la taille

```
fenetre.update_idletasks()
fenetre.geometry("")           # Ajuste automatiquement la taille
fenetre.minsize(width, height) # Fixe une taille minimale
```

Parfait pour adapter la fenêtre après un changement de langue ou de texte.

4) Internationalisation (i18n)

Structure conseillée

```
/lang/
└── lang_fr.json
└── lang_en.json
└── lang_es.json
```

Chaque fichier contient les traductions pour une langue spécifique.

Exemple lang_fr.json

```
{
    "titre": "Convertisseur de bases numériques",
    "btn_convertir": "Convertir",
    "btn_effacer": "Effacer",
    "btn_quitter": "Quitter",
    "menu_fichier": "Fichier",
    "menu_aide": "Aide",
    "menu_langue": "Langue"
}
```

Exemple lang_en.json

```
{
    "titre": "Numeric base converter",
    "btn_convertir": "Convert",
    "btn_effacer": "Clear",
    "btn_quitter": "Quit",
    "menu_fichier": "File",
    "menu_aide": "Help",
    "menu_langue": "Language"
}
```

Chargement des fichiers de langue

```
import json

def charger_traductions(fichier):
    try:
        with open(fichier, "r", encoding="utf-8") as f:
            return json.load(f)
    except FileNotFoundError:
        return {"titre": "Erreur : fichier de langue introuvable"}
```

💡 Mise à jour de l'interface

```
def mettre_a_jour_interface():
    fenetre.title(textes_langues["titre"])
    bouton_convertir.config(text=textes_langues["btn_convertir"])
    bouton_effacer.config(text=textes_langues["btn_effacer"])
    bouton_quitter.config(text=textes_langues["btn_quitter"])
```

Changement de langue à la volée

```
def changer_langue(nouvelle_langue):
    global langue_actuelle, textes_langues
    langue_actuelle = nouvelle_langue
    nom_fichier = f"lang_{nouvelle_langue}.json"
    textes_langues = charger_traductions(nom_fichier)
    mettre_a_jour_interface()
    fenetre.update_idletasks()
    fenetre.geometry("")
```

Astuce : pour un affichage immédiat, placer cette fonction dans le `command=` des boutons drapeaux.

Bonnes pratiques i18n

- Stocker les traductions dans des fichiers `.json` séparés.
 - Encoder les fichiers en **UTF-8** (gestion des accents).
 - Toujours appeler `update_idletasks()` après un changement de langue.
 - Éviter les chaînes "en dur" dans le code — passer systématiquement par `textes_langues["clé"]`.
 - Créer des clés explicites et homogènes (`btn_*`, `menu_*`, `titre_*`).
-

Exemple minimal complet

```
import tkinter as tk, json

# Chargement par défaut
def charger_traductions(fichier):
    with open(fichier, "r", encoding="utf-8") as f:
        return json.load(f)

textes_langues = charger_traductions("lang_fr.json")

# Fenêtre
fenetre = tk.Tk()
fenetre.title(textes_langues["titre"])

# Widgets
label = tk.Label(fenetre, text=textes_langues["titre"])
bouton_convertir = tk.Button(fenetre, text=textes_langues["btn_convertir"])
label.pack(pady=10)
bouton_convertir.pack(pady=5)

# Fonction de changement de langue
def changer_langue(nouvelle_langue):
    global textes_langues
    textes_langues = charger_traductions(f"lang_{nouvelle_langue}.json")
    fenetre.title(textes_langues["titre"])
    label.config(text=textes_langues["titre"])
    bouton_convertir.config(text=textes_langues["btn_convertir"])

# Boutons de langue
tk.Button(fenetre, text="FR", command=lambda:
    changer_langue("fr")).pack(side="left", padx=10)
tk.Button(fenetre, text="EN", command=lambda:
    changer_langue("en")).pack(side="left", padx=10)

fenetre.mainloop()
```

Checklist finale

- Créer un dossier `lang/` pour tes fichiers `.json`

- Charger la langue au lancement via `charger_traductions()`
 - Mettre à jour les textes via `mettre_a_jour_interface()`
 - Ajouter des boutons ou un menu pour changer la langue dynamiquement
 - Vérifier la compatibilité des caractères accentués (UTF-8)
 - Ne laisser aucun texte codé en dur dans l'interface
-

Qui veut une Chimay rouge ? 🍺 😊