Documentation Python - Projet Stage

Création d'une application d'exercices de systèmes et codage de l'information

Auteurs : Kone Yacouba ,Akoury Joseph, DIOP Amadou Tidiane, DISSOU Kefa, IRANNEZAHAD Avin, KAFABA Alimou, TENDOMENE Ora, VERDIER Liam.

Année : 2022/2023

SOMMAIRE

➔ Configuration de la taille des lignes et des colonnes

➔ Redimensionnement de la fenêtre principale

➔ Outil grid pour positionner les widgets

➔ Variable Global

➔ Utilisation des Entry

➔ Fonction Lambda

➔ Fonction Sorted

➔ Canvas pour la création des cases

➔ Créer un exécutable

➔ Configuration de la taille des lignes et des colonnes

Lien utile : https://stackoverflow.com/questions/45847313/what-does-weight-do-in-tkinter

Syntaxe : ***widget.rowconfigure(N, weight)***
widget.columnconfigure(N, weight)

N : numéro de ligne ou de colonne
Weight : Échelle relative pour répartir l'espace supplémentaire.

Dans les termes les plus simples possibles, un **weight (poids) non nul** agrandit une ligne ou une colonne s'il y a encore de l'espace. La valeur par défaut est un poids de zéro, ce qui signifie que la colonne ne grandira pas.

➢ Considérez le code suivant, qui crée une fenêtre plus grande que les widgets qui s'y trouvent, et pour laquelle aucune colonne n'a de poids :

```
from tkinter import *
fenetre = Tk()
fenetre.title('Documentation_python')
fenetre.geometry('200x100')

f1 = Frame(fenetre, background="bisque", width = 10, height = 100)
f2 = Frame(fenetre, background="pink", width = 10, height = 100)

f1.grid(row=0, column=0)
f2.grid(row=0, column=1)

fenetre.grid_columnconfigure(0,weight=0)
fenetre.grid_columnconfigure(1,weight=0)

fenetre.mainloop()
```

Figure 1 - a : code explicatif de la configuration d'une fenêtre tkinter

Tkinter a été informé de ne donner aucun espace supplémentaire à aucune des colonnes, de sorte que l'espace supplémentaire reste inutilisé à droite. Figure I-b

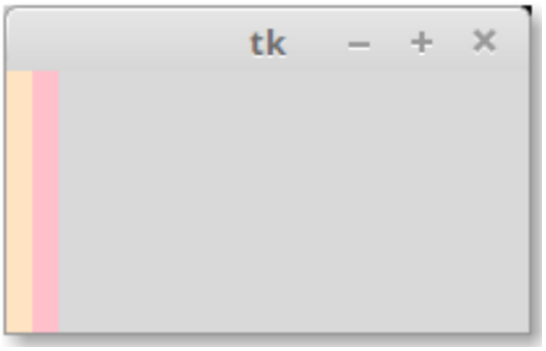


Figure 1 - b : la fenêtre que l'on obtient à partir du code de la Figure 1 - a

➢ Maintenant, si on change le code en donnant un poids à une seule colonne

fenetre.grid_columnconfigure(0, weight=1)

Parce que **la colonne 0** maintenant à **(weight=1)**, tkinter donne l'espace supplémentaire à cette colonne.

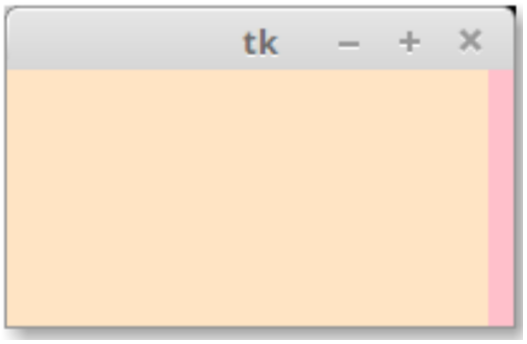


Figure 1 - c : la fenêtre obtenue

- ➢ Enfin, si on essaye de donner un poids aux deux colonnes, l'espace supplémentaire sera réparti entre les colonnes proportionnellement en fonction de leurs poids.

Par exemple, supposons qu'on souhaite avoir une zone de navigation sur la gauche qui occupe 1/4 de l'écran et une zone principale qui occupe les 3/4 de l'écran.

fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=3)

Les deux colonnes ont un poids, donc, un espace supplémentaire est attribué aux deux colonnes. Pour chaque quatre pixels d'espace supplémentaire, **la colonne 0 recevra 1 et la colonne 1 recevra les 3 autres**.

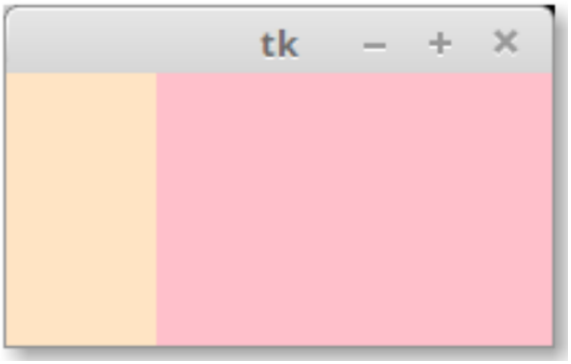


Figure 1 - d : la fenêtre obtenue

Afin de pouvoir redimensionner la fenêtre verticalement, on utilise **rowconfigure** en se basant sur le même principe et techniques de gestion de la taille colonnes décrites dans la section précédente.

➔ Outil grid pour positionner les widgets

Lien utile : http://tkinter.fdex.eu/doc/gp.html

Syntaxe : ***widget.grid(row=x, column=y,columnspan=z, sticky = “”)***

Exemple :

Sur la Figure 2 - a, on distingue 3 boutons placés l’un à côté de l’autre sur la ligne 0, ainsi qu’un seul bouton *“Bouton Test”* placé sur la ligne 1.

Si on veut que notre bouton *“Bouton Test”* soit centré, l’option *“columnspan = 3”* (Figure 2 - c)va venir fusionner les colonnes jusqu’à la colonne 3, mais uniquement sur la ligne propre à ce bouton - la ligne 1 comme sur la Figure 2 - b.

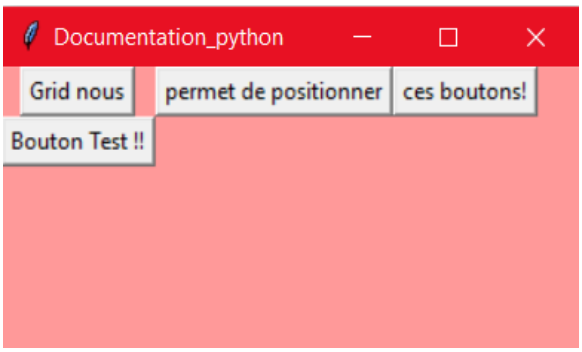


Figure 2 - a : sans columnspan

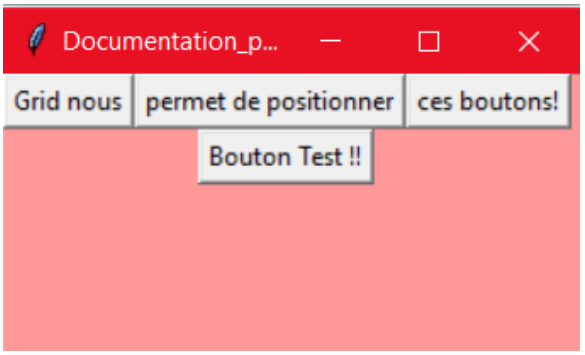


Figure 2 - b : avec columnspan = 3

```
from tkinter import *
fenetre = Tk()
fenetre.title('Documentation_python')
fenetre.geometry('400x300')
fenetre.config(bg='#ff9999')

Button(fenetre, text="Grid nous").grid(row=0, column=0)
Button(fenetre, text="permet de positionner").grid(row=0, column=1)
Button(fenetre, text="ces boutons!").grid(row=0, column=2)

Button(fenetre, text="Bouton Test !!").grid(row=1, column =0, columnspan=3)

fenetre.mainloop()
```

Figure 2 - c : code explicatif de la gestion de positionnement par .grid

On peut ajouter d’autres options pour positionner correctement le texte de notre choix:

- ***sticky*** = va venir aligner, centrer, étirer

  sticky='w' ou **'e'** ou **'se'** ou **'nw'** (cela centre vers la gauche, la droite, le sud gauche, l e nord droit… cela correspond aux points cardinaux en anglais

  sticky='ns' étire verticalement tout en le laissant centré horizontalement.

  sticky='ew' étire horizontalement tout en le laissant centré verticalement.

  sticky='nesw' étire dans les deux directions afin de remplir la cellule.

- ***Ipady et ipadx*** : marge interne verticale (en y) ou horizontale (x). Cette dimension est ajoutée à l'intérieur du widget sur ses côtés haut et bas.

➔ Variable Global

Syntaxe: ***global Nom_Variable***

Dans une fonction, les variables sont locales pour les utiliser en dehors de leur fonction on utilise global.
Par exemple, sur la figure suivante, titre est une variable globale celà veut dire que l'on peut l'utiliser ou la modifier en dehors de la fonction *"ChapChoix"*.

```
48    def ChapChoix():
49        global titre
50        global soustitre
51        global signe
52        global conversion
53        global RetourM
```

Figure 3 : Exemple d'utilisation du global dans notre code

➔ Utilisation des Entry :

Lien utile : Entry - Champs de saisie — Tkinter pour ISN (fdex.eu) (Autoformation)
Syntaxe : ***widget= entry(master,option…)***

Cet outil permet d'utiliser les données saisies par un utilisateur. Entry peut avoir plusieurs paramètres représentant des options tels que la taille, la couleur ou une commande. (voir lien)

Les paramètres :

Option : bg,bd,command, state
Master : fenêtre parents

Les méthodes :

- **Get()** : permet de récupérer le contenu d'un entry sous forme de string.
On peut également utiliser var.Cget('option') » qui retourne la valeur d'une option en string ou l'option d' un widget.

- **Delete(pos)** : permet de supprimer les caractères dans un entry
Exemple : *delete(0,END)*. Supprime tout le contenu de l'entry.

- **Insert (pos,s)** : permet d'insérer un string à une certaine position.

- **Configure(state)** : permet de configurer les options d'une ou plusieurs options. Utiliser notamment pour bloquer un case entry.

```
Nb.configure(state="normal")
Nb.delete(0,END)
Nb.insert(0,"Donnée inutile")
Nb.configure(state="disabled")
```

Figure 4: fonctions des entrys

➔ Les Message Box:

Lien utile : [tkinter.messagebox — Tkinter message prompts — Python 3.10.5 documentation]

Les messagebox permettent de faire apparaître des messages sous forme de pop-up.
Il existe plusieurs types de messages : information, erreurs, alerte ou encore sous formes de questions.

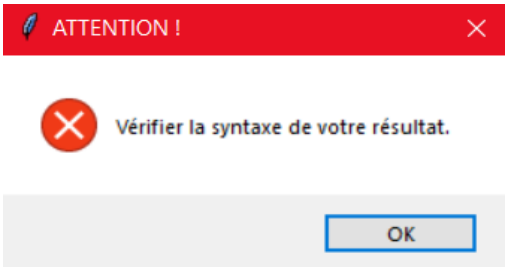
Info :
- ➔ messagebox.showinfo(title=None, message=None, **options)

Alerte :
- ➔ messagebox.showwarning(title=, message=None, **options)
- ➔ messagebox.showerror(title=None, message=None, **options)

Question :
- ➔ messagebox.askquestion(title=None, message=None, **options)
- ➔ messagebox.askokcancel(title=None, message=None, **options)
- ➔ messagebox.askretrycancel(title=None, message=None, **options)
- ➔ messagebox.askyesno(title=None, message=None, **options)
- ➔ messagebox.askyesnocancel(title=None, message=None, **options)

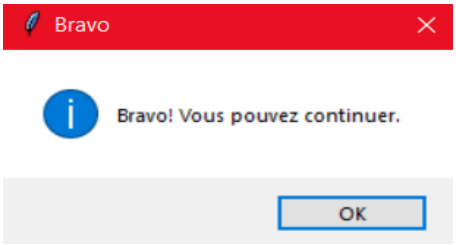
- **Message box qui affiche une erreur/warning :**

messagebox.showerror*("ATTENTION !", "Vérifier la syntaxe de votre résultat.")*



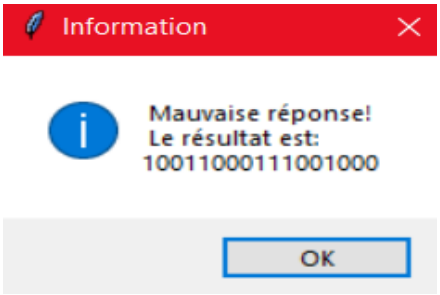
- **Message box qui affiche un message quelconque :**

messagebox.showinfo*("Bravo", "Bravo! Vous pouvez continuer")*



➔ **Message box qui affiche une variable incluse :**
On suppose x une variable tel que : x = 10011000111001000

messagebox.showinfo*("Information", "Mauvaise réponse! \n Le résultat est: \n"+* ***("".join(x))***



➔ Fonction Lambda dans les commandes :

Lien utile :
https://python.developpez.com/cours/DiveIntoPython/php/frdiveintopython/power_of_introspection/lambda_functions.php

Syntaxe : ***lambda x : y*** *(x est une variable et y est une expression)*

La fonction lambda est une fonction anonyme contenant une seule expression. Elles sont très utiles lorsqu'il faut effectuer des petites tâches avec moins de code.

```
>>> def f(x):
        return x*2

>>> f(3)
6
>>> g = lambda x : x*2
>>> g(3)
6
```

Dans notre cas, on l'a utilisé un peu différemment et surtout lorsque on voulait exprimer une fonction qui prend comme argument une autre fonction. Le but c'était de pouvoir passer d'une fonction à une autre, et éviter le problème d'exécution de plusieurs fonctions en même temps:

```
B3=Button(fen, text="Valider", font=("courier", 18, "italic"),
          fg='white', bg='#103985', width=15, height=2, command=lambda:(get(basedep,basearr,entier,man)))
```

```
if Type=="PCTER":
    pret=sorted(pret, key=lambda x: x["d"])
if Type=="Priorite fixes":
    pret=sorted(pret, key=lambda x: x["prio"])
```

➔ Fonction Sorted :

Syntaxe : ***sorted (itérable[, cmp[, key[, reverse]]])***

Renvoie une liste triée à partir d'itérable.
L'itérable est requis.

cmp (Facultatif):
C'est une fonction de comparaison personnalisée de deux arguments (éléments itérables).

Key (Facultatif):
Une fonction à un argument qui est utilisée pour extraire une clé de comparaison de chaque élément de la liste. La valeur par défaut est None (compare les éléments directement).

Reverse (Facultatif):
Nous permet de trier les éléments de la liste selon l'ordre inverse.

➔ Canvas pour la création des cases :

Lien utile : https://infoforall.fr/python/python-act130.html#partie_04

Commençons par créer la frame qui va nous permettre de positionner le Canvas.

frame_main = tk.Frame(fenetre)

frame_canvas = tk.Frame(frame_main)

➢ Ensuite, création du widget que nous allons utiliser pour dessiner : le Canvas.canvas = ***tk.Canvas(frame_canvas).***

On utilise pack, place (x, y) ou grid (row, column) pour placer le début du Canvas au coordonnées choisis.

➢ Enfin, création des boutons à l'intérieur du Canvas.

frame_buttons = tk.Frame(canvas)

canvas.create_window((0, 0), window=frame_buttons, anchor='nw')

Processus	Arrivée	Durée
p1		
p2		
p3		
p4		
p5		
p6		
p7		
p8		
p9		
p10		

(0,0) sont les coordonnées respectives x et y d'origine.

➔ Créer un exécutable:

Liens utiles:

➔ https://wiki-fablab.grandbesancon.fr/doku.php?id=howto:python:pyinstaller

➔ https://pyinstaller.org/en/stable/?fbclid=IwAR2oqYyYiJV9D9SG8YWjPYCkSL7YtdTJzIB rkeoQFNs_tA1ypzf8_msr20

L'explication pour l'exécutable est sous Windows, pour linux c'est légèrement différent au début (voir la documentation)

Sous Windows:

Premièrement étape , il faut se placer dans le dossier suivant :

C:\Program Files\Pyton\PythonX.X\Scripts

Où (Si python n'est pas installé en administrateur)

C:\Users\votre_nom_utilisateur\AppData\Roaming\Python\PythonX.X\Scripts

Petite astuce passer par windows+R puis lancer %appdata% puis cherche python il se peut qu'il ne soit pas dans Roaming mais dans local

Deuxième étape installer PIP

• Vérifier s'il est installé par = *pip -v*

- S'il n'est pas installé , rentrer cette commande : *python get-pip.py*
- Mettre à jour pip = *pip install –upgrade pip*

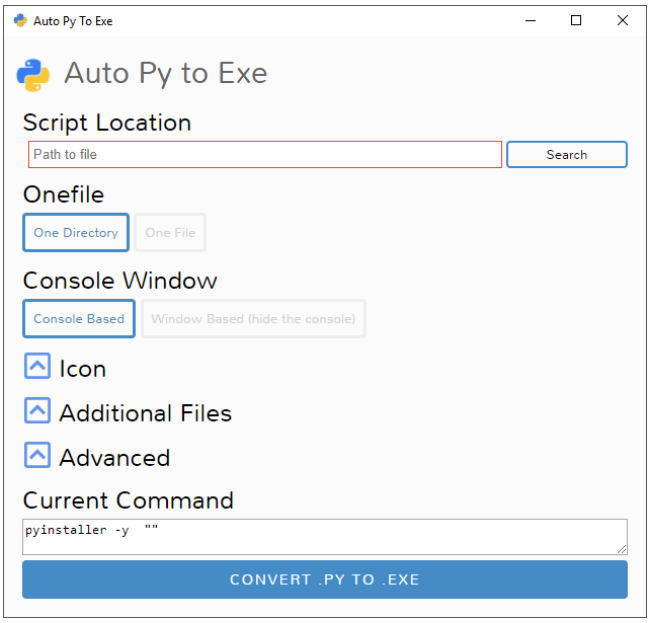
Troisième étape installation de Pyinstaller

pip install -U pyinstaller

Ensuite pour créer l'exécutable on va installer auto-py-to-exe:

pip install auto-py-to-exe

puis on exécute avec *auto-py-to-exe* cette fenêtre devrait s'ouvrir :



Debuggage d'une application d'exercices de systèmes et codage de l'information et ajout de fonctionnalité

Auteurs :

Année : 2022/2023

changement des variable et des fonctions

Nous avons constaté dans la dernière version de l'application que de nombreux noms de variables n'étaient pas suffisamment explicites. Cependant, nous avons pris l'initiative de les renommer de manière à les rendre plus significatifs et parlants.

En effectuant ces changements, nous avons veillé à choisir des noms de variables et de fonctions qui reflètent clairement leur objectif et leur utilisation dans le contexte de l'application.

Renommer les variables et les fonctions de manière descriptive améliore la lisibilité et la compréhension du code, tant pour nous que pour les autres développeurs qui pourraient travailler sur le projet.

TopLevel

Les Toplevels sont des fenêtres indépendantes créées dans une interface graphique. Ils sont créés à partir d'une fenêtre parente existante, généralement en utilisant la méthode Toplevel() de la bibliothèque graphique . Elles sont des fenêtres indépendantes liées à une fenêtre parente. Dans le cadre de notre stage, cela nous permet de découper le code en différents modules et d'afficher uniquement les modules souhaités.

Voici un exemple de la fenêtre toplevel de *fenetreCodage*, qui est l'enfant de la fenêtre principale *fenetreMenu.*

```
def open_codage(fenetreMenu,x,y):
    fenetreCodage = Toplevel(fenetreMenu)
```

Voici la hiérarchie des fenêtres d'application :

1) Codage :

fenetreMenu (fenêtre principale) -> fenetreCodage (fenêtre enfant de fenetreMenu -> fenetrePchoix (fenêtre enfant de fenetreCodage -> Les fenetreExo X (les enfant de fenetrePchoix)

2) Ordonnancement :

fenetreMenu (fenêtre principale) -> fenetreOrdo -> fenetreOrdoChoix -> fenetreExo-prdo

3) Gestion de la mémoire :

????

En plus de l'outil TopLevel, nous souhaitons masquer la fenêtre précédente afin de ne voir qu'une seule fenêtre. Pour ce faire, nous avons utilisé la fonction :

```
def open_exercice(fenetreCodage,fenetreMenu,nom_exo):
    fenetreCodage.withdraw()  # Masquer l'interface 1
    interface_pageC.open_exercice(fenetreCodage,fenetreMenu,nom_exo)
```

Qui nous permet d'ouvrir que l'exercice en question, le ".withdraw()" sert à masquer la précédente fenêtre.

Nous souhaitons également faire apparaître la fenêtre précédente lorsque nous fermons la nouvelle fenêtre, pour cela, nous utiliserons la fonction :

```
def retour_menu(fenetreMenu,fenetreCodage):
    fenetreCodage.destroy()
    fenetreMenu.deiconify()  # Afficher l'interface 1
```

Enfin, afin de faire appel aux différents modules dans notre programme, nous avons procédé comme suit :

```
import fonction_chap1
import interfaceExo1
import interfaceExo2
import interfaceExo3
import interfaceExo4
import interfaceExo5
import InterfaceExo6
import InterfaceExo7
import InterfaceExo8
```

Redimensionnement de la fenêtre

```
def  open_Exo(fenetreOrdoChoix,select,fenetreMenu,nom_exo):

    fenetreExo_ordo=Toplevel(fenetreOrdoChoix)

    fenetreExo_ordo.config(background="#b0e2ff")

    fenetreExo_ordo.title("Application Python")

    parent_width = fenetreOrdoChoix.winfo_width()

    parent_height = fenetreOrdoChoix.winfo_height()

    screen_width = fenetreExo_ordo.winfo_screenwidth()

    screen_height = fenetreExo_ordo.winfo_screenheight()

    window_width = int(screen_width * 0.6)

    window_height = int(screen_height * 0.6)

    x = (screen_width - window_width) // 2

    y = (screen_height - window_height) // 2

    fenetreExo_ordo.geometry(f"{parent_width}x{parent_height}+{x}+{y}")

    fenetreExo_ordo.minsize((window_width+150), (window_height+150))
```

La fenêtre d'exercice (fenetreExo_ordo) est configurée avec une couleur de fond spécifique (#b0e2ff) et un titre ("Application Python"). Les dimensions de la fenêtr.0e parente (fenetreOrdoChoix) sont récupérées à l'aide des méthodes winfo_width() et winfo_height().

Ensuite, les dimensions de l'écran sont récupérées à l'aide des méthodes winfo_screenwidth() et winfo_screenheight() de la fenêtre d'exercice (fenetreExo_ordo). Ces dimensions sont utilisées pour calculer les dimensions de la fenêtre d'exercice en fonction d'une proportion spécifique (60% de la largeur et de la hauteur de l'écran).

La position de la fenêtre d'exercice (fenetreExo_ordo) est calculée en fonction des dimensions de l'écran et de la fenêtre parente. Cela permet de centrer la fenêtre d'exercice sur l'écran.

Enfin, la géométrie de la fenêtre d'exercice est définie à l'aide de la méthode geometry() en utilisant les dimensions calculées précédemment. De plus, une taille minimale est définie pour la fenêtre d'exercice à l'aide de la méthode minsize() pour garantir qu'elle a une taille suffisante pour afficher correctement le contenu.

Ajuster la couleur en fonction du système d'exploitation

Nous avons utilisé une conditionnelle en utilisant une structure "si...sinon" pour définir les valeurs des variables couleur_bouton (couleur du bouton) et couleur_texte (couleur du texte) en fonction de la valeur de my_os (système d'exploitation).

```
my_os = platform.system()
```

La fonction platform.system() renvoie une chaîne de caractères qui représente le nom du système d'exploitation.

```
if my_os == "Linux" or my_os == "Windows":
        couleur_bouton = "#1e90ff"
        couleur_texte = "#ffffff"
    else:
        couleur_bouton = "white"
        couleur_texte = "#000000"
```

Association d'événements à des fonctions de changement de couleur

En utilisant bind, nous avons défini des actions spécifiques à exécuter en réponse à des événements tels que le passage de la souris sur un bouton ('<Enter>') ou le retrait de la souris de la zone du bouton ('<Leave>'). Cela permet de personnaliser le comportement de notre interface graphique en fonction des interactions de l'utilisateur.

```
```

```
    Bexo1.bind('<Enter>', lambda event: change_couleur(event,
'#7ACFB0'))  # Quand la souris entre dans la zone du bouton

    Bexo1.bind('<Leave>', restaurer_couleur)  # Quand la
souris quitte la zone du bouton
```

Ajustement de la taille et de la police en fonction de la fenêtre

Pour ajuster la taille de chaque fenêtre, le positionnement des boutons et la police sur chaque ordinateur, nous avons utilisé les fonctions suivantes : ajuster_taille, ajuster_bouton et maj_font_size.

La fonction ajuster_taille est utilisée pour adapter la taille de chaque fenêtre en fonction des spécifications de l'ordinateur sur lequel l'application est exécutée.La fonction ajuster_bouton est utilisée pour ajuster le positionnement des boutons dans chaque fenêtre, en tenant compte des dimensions de la fenêtre et des préférences de disposition. Enfin, la fonction maj_font_size est utilisée pour mettre à jour la taille de la police utilisée dans chaque fenêtre, en fonction de la résolution de l'écran et des préférences de l'utilisateur. Cela garantit que le texte est lisible et adapté à chaque ordinateur.

```
 def ajuster_taille(event):

    # Redimensionnement des boutons en fonction de la taille de la
fenêtre

        largeur_fenetre = fenetreExo6.winfo_width()

        hauteur_fenetre = fenetreExo6.winfo_height()



        label_font_size = int(largeur_fenetre / 70)

        entry_font_size = int(largeur_fenetre / 90)

        tIndication_font_size = int(largeur_fenetre / 110)

        titre=int(largeur_fenetre/50)

        tIndication.config(font=("courier", tIndication_font_size,
"bold"))

        titre.config(font=("courier",titre,"bold"))
```

```
        tConversion.config(font=("courier", label_font_size, "bold"))

        tFormat.config(font=("courier", label_font_size, "bold"))

        tVirgule.config(font=("courier", label_font_size, "bold"))

        tResultat.config(font=("courier", label_font_size, "bold"))
```

```
def ajuster_bouton(event):
```

```
        largeur_fenetre = fenetreExo6.winfo_width()

        hauteur_fenetre = fenetreExo6.winfo_height()

        bouton_width = int(largeur_fenetre / 50)

        bouton_height = int(hauteur_fenetre / 350)



        bRappel.config(width=bouton_width, height=bouton_height)

        bNouveau.config(width=bouton_width, height=bouton_height)

        bValider.config(width=bouton_width, height=bouton_height)

        bMenu.config(width=bouton_width, height=bouton_height)

        bQuitter.config(width=bouton_width, height=bouton_height)

   def maj_font_size(text_widget,nb,event):
        # Mettre à jour la taille de police en fonction de la largeur
de la fenêtre
        font_size = int(nb * fenetreExo6.winfo_width() / 1000)
        text_widget.config(font=("Courier", font_size, "bold"))
```

La fonction "restaurer_couleur" change la couleur de fond et la couleur d'arrière-plan actif d'un widget pour les définir sur "#1e90ff". Cela permet de restaurer visuellement la couleur du widget lorsqu'un événement se produit.

```
def restaurer_couleur(event):
    event.widget.config(bg="#1e90ff", activebackground="#1e90ff")  #
Restaurer la couleur d'origine du bouton lors du survol
```