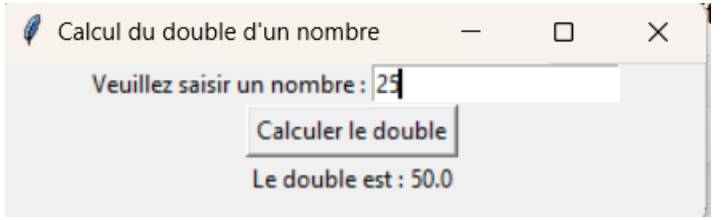
TDs avec TKInter

Dr. NSENGE MPIA HERITIER, PhD

TD 1: Exemple d'entrée/sortie

- Ce code lit un nombre et retourne son double

```python
import tkinter as tk
#Création de la méthode qui se charge de calcul
def calculate_double():
    try:
        # Récupérer le nombre saisi par l'utilisateur
        number = float(entry.get())
        # Calculer le double du nombre
        result = number * 2
        # Mettre à jour le label avec le résultat
        #On configure le texte du label result_label pour afficher "Le double est : " suivi de la valeur calculée result.
        #{result} c'est pour formater l'affichage
        result_label.config(text=f"Le double est : {result}")
    except ValueError:
        # Afficher un message d'erreur si la saisie n'est pas un nombre
        result_label.config(text="Veuillez saisir un nombre valide.")
# Créer une fenêtre
root = tk.Tk()
root.title("Calcul du double d'un nombre")
# Créer un cadre pour contenir le message et la zone de saisie
frame = tk.Frame(root)
frame.pack()
# Créer un label pour le message avant la saisie du nombre
message_label = tk.Label(frame, text="Veuillez saisir un nombre :")
message_label.pack(side=tk.LEFT)
# Créer une zone de saisie avec un placeholder
entry_placeholder = "Entrez un nombre"
#Cette ligne crée un widget de saisie Entry et le place dans le cadre frame.
entry = tk.Entry(frame)
#Cette ligne insère un texte par défaut (ou placeholder) dans le champ de saisie.
#Le texte par défaut est défini comme entry_placeholder ci-haut, qui est une chaîne de caractères.
#Le 0 représente l'indice où le texte entry_placeholder doit être inséré dans le widget entry.
#Dans un widget de saisie Entry de Tkinter, les caractères sont indexés à partir de 0, de sorte que 0 représente
#la position avant le premier caractère.
#En insérant du texte à l'indice 0, vous placez le texte au tout début du champ de saisie.
#Cela est couramment utilisé pour afficher un texte par défaut ou un placeholder dans le champ de saisie,
#indiquant à l'utilisateur ce qu'il doit saisir.
entry.insert(0, entry_placeholder)
# Cette ligne place le widget entry à gauche des autres widgets dans le cadre frame.
#Le paramètre side=tk.LEFT spécifie que le widget doit être placé du côté gauche du cadre.
#Si vous avez d'autres widgets dans le cadre et que vous souhaitez les placer différemment,
#vous pouvez utiliser side=tk.TOP, side=tk.RIGHT ou side=tk.BOTTOM pour modifier la position du widget.
entry.pack(side=tk.LEFT)
# Créer un bouton pour calculer le double
calculate_button = tk.Button(root, text="Calculer le double", command=calculate_double)
#Tkinter organise automatiquement le widget du button dans la fenêtre principale en fonction des autres widgets déjà présents
calculate_button.pack()
result_label = tk.Label(root, text="")# Créer un label pour afficher le résultat
#Tkinter organise automatiquement le widget du label dans la fenêtre principale en fonction des autres widgets déjà présents
result_label.pack()
root.mainloop() # Démarrer la boucle principale
```



TD 2: Exemple Equation du second degré

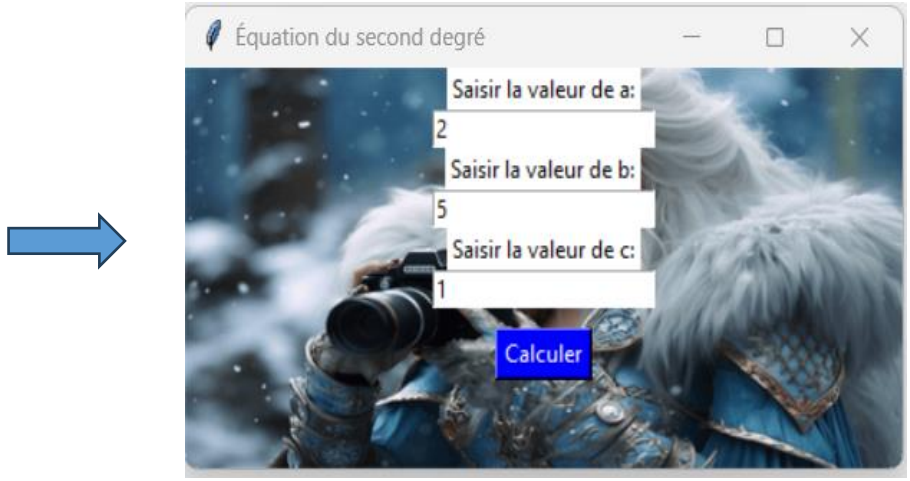
```python
import tkinter as tk
from tkinter import messagebox
import math
def on_calculer():
    try:
        a = float(entry_a.get())
        b = float(entry_b.get())
        c = float(entry_c.get())
        # Calcul du discriminant
        discriminant = b**2 - 4*a*c
        # Si le discriminant est négatif, il n'y a pas de solution réelle
        if discriminant < 0:
            messagebox.showinfo("Résultat", "L'équation n'a pas de solution réelle.")
        # Calcul des solutions
        elif discriminant == 0:
            # Calcul de la solution unique
            x = -b / (2*a)
            messagebox.showinfo("Résultat", f"L'équation a une racine double x1=x2={x}")
        else:
            x1 = (-b + math.sqrt(discriminant)) / (2*a)
            x2 = (-b - math.sqrt(discriminant)) / (2*a)
            messagebox.showinfo("Résultat", f"Les solutions de l'équation sont x1={x1} et x2={x2}")
    except ValueError:
        messagebox.showerror("Erreur", "Veuillez entrer des coefficients valides.")
# Création de la fenêtre principale
root = tk.Tk()
root.title("Équation du second degré")
# Centrer la fenêtre au milieu de l'écran
window_width = 400
window_height = 200
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x_position = (screen_width - window_width) // 2
y_position = (screen_height - window_height) // 2
root.geometry(f"{window_width}x{window_height}+{x_position}+{y_position}")
# Ajouter une image de fond à la fenêtre principale
background_image = tk.PhotoImage(file="C:\\Users\\Staniher\\Pictures\\compress_image.png")
background_label = tk.Label(root, image=background_image)
background_label.place(relwidth=1, relheight=1)
# Entrées pour les coefficients a, b, et c
label_a = tk.Label(root, text="Saisir la valeur de a:", bg="white")
label_a.pack()
entry_a = tk.Entry(root)
entry_a.pack()

label_b = tk.Label(root, text="Saisir la valeur de b:", bg="white")
label_b.pack()
entry_b = tk.Entry(root)
entry_b.pack()

label_c = tk.Label(root, text="Saisir la valeur de c:", bg="white")
label_c.pack()
entry_c = tk.Entry(root)
entry_c.pack()

# Ajouter une couleur de fond au bouton "Calculer"
button_calculer = tk.Button(root, text="Calculer", command=on_calculer, bg="blue", fg="white")
button_calculer.pack(pady=10)

# Boucle principale Tkinter
root.mainloop()
```



Création d'Exécutable

- Pour créer un exécutable d'une application Tkinter, vous pouvez utiliser des outils comme PyInstaller ou cx_Freeze.
- Voici comment procéder avec PyInstaller :
  - Assurez-vous que votre application Tkinter fonctionne correctement.
  - Installez **PyInstaller** si ce n'est pas déjà fait : *pip install pyinstaller*.
  - À l'invite de commande de Anaconda, naviguez jusqu'au répertoire contenant votre script Tkinter.
  - Utilisez PyInstaller pour créer l'exécutable en saisissant par exemple:

    pyinstaller --onefile SecondDegre.py
  - PyInstaller va créer un dossier dist dans le répertoire de votre script, contenant l'exécutable de votre application.

Exercice avec BDD

Exemple de l’artéfact à construire

Gestion des profiles des étudiants

- Base de données: ***profile_etudiants***

```sql
CREATE TABLE IF NOT EXISTS `etudiants` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `image_profile` mediumtext,
  `matricule` mediumtext,
  `nom` mediumtext,
  `postnom` mediumtext,
  `phone` mediumtext,
  `email` mediumtext,
  `addresse` mediumtext,
  `date_mise_a_jour` mediumtext,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=7 ;
```

Gestion des profiles des étudiants (Cont.)

- Début du code python

```python
#On doit installer pip install pymysql
import pymysql #On importe le module PyMySQL pour interagir avec une base de données MySQL
import tkinter as tk #On importe le module tkinter sous l'alias tk pour créer une interface graphique.
import os #On importe le module os pour accéder aux fonctionnalités liées au système d'exploitation, comme la gestion des fichiers.
import random #On importe le module random pour générer des nombres aléatoires. On l'utilisera pour générer des nom des images des profiles
import string #On importe le module string pour accéder à des constantes de chaînes de caractères comme ascii_lowercase
import csv #On importe le module csv pour lire et écrire des fichiers CSV. On l'utilisera pour sauvegarder nos données en CSV
#On importe les classes date et datetime du module datetime pour gérer les dates et les heures. On va commencer à enregistrer la date actuelle
from datetime import date,datetime
#On importe le module ttk de tkinter pour créer des widgets améliorés.
from tkinter import ttk
#pip install pillow pour installer PIL
from PIL import Image #On importe le module Image du package PIL (Pillow) pour manipuler des images.
#On importe divers widgets et fonctionnalités de tkinter pour la création d'une interface graphique.
#Canvas est un widget tkinter qui sert de zone de dessin pour dessiner des formes, du texte, des images, etc.
#Vous pouvez utiliser un Canvas pour créer des graphiques simples,
#des diagrammes, des animations, etc. Il vous permet de contrôler les coordonnées des objets dessinés et d'interagir avec eux.
#Tk nous aidera à créer une nouvelle fenêtre
## Tk crée une nouvelle fenêtre Tkinter sans utiliser d'alias (ex. import tkinter as tk
from tkinter import Tk, Canvas, Button, PhotoImage, messagebox, Entry, filedialog

#On définit une fonction qui retourne la date actuelle au format "mois/jour/année".
def getDateActuelle():
    return f"{date.today().month}/{date.today().day}/{date.today().year}"

#On génère une chaîne de caractères aléatoires de longueur length (12 par défaut) en utilisant les lettres minuscules de l'alphabet.
def generer_caracteres_aleatoires(length=12):
    caracteres = string.ascii_lowercase
    caracteres_aleatoires = ''.join(random.choice(caracteres) for _ in range(length))
    return caracteres_aleatoires
```

Gestion des profiles des étudiants (Cont.)

- Code pour la connexion à la BDD toujours dans le même fichier

```python
#On crée une fonction qui va nous permettre de nous connecter au serveur BDD
def connection():
    conn=pymysql.connect(
        host='localhost',
        user='root',
        password='',
        db='profile_etudiants'
    )
    return conn
```

Gestion des profiles des étudiants (Cont.)

- Code pour exporter le contenu de la BDD dans un fichier excel

```
#Ce code Python définit une fonction exporterExcel() qui exporte les données
#de la table etudiants se trouvant dans notre base de données MySQL vers un fichier CSV
def exporterExcel():
    conn=connection() #On appelle la fonction connection() déclarée dans le slide précédent qui renvoie la connexion à une base de données.
    #On crée un objet curseur à partir de la connexion à la base de données.
    #Le curseur est utilisé pour exécuter des requêtes SQL et récupérer les résultats.
    cursor=conn.cursor()
    #On vérifie si la connexion à la base de données est active en envoyant une requête ping.
    #Cela peut être utile pour éviter les déconnexions inattendues.
    cursor.connection.ping()
    #On définit une requête SQL pour sélectionner toutes les lignes de la table etudiants et les ordonner par id de manière décroissante.
    sql=f"SELECT * FROM etudiants ORDER BY `id` DESC"
    #On exécute la requête SQL pour récupérer les données de la table etudiants.
    cursor.execute(sql)
    #On récupère toutes les lignes de résultats de la requête et les stocke dans la variable donneesBrutes.
    donneesBrutes=cursor.fetchall()
    date = str(datetime.now())  #On obtient la date et l'heure actuelles sous forme de chaîne de caractères.
    #Dans les trois lignes qui suivent, on nettoie la chaîne de caractères de la date pour la rendre compatible avec un nom de fichier,
    #en remplaçant les espaces par des underscores et les deux-points par des tirets.
    date = date.replace(' ', '_')
    date = date.replace(':', '-')
    #On fait l'opération de découpage de chaîne (slicing).
    #On prend les 16 premiers caractères ([début:fin]) de la chaîne date et les assigne à la variable dateFinal.
    dateFinal = date[0:16]
    #On ouvre un fichier CSV en mode ajout ('a') pour ajouter des données. Le nom du fichier est basé sur la date et l'heure actuelles.
    with open("etudiants_"+dateFinal+".csv",'a',newline='') as f:
        #On crée un objet écrivain CSV pour écrire dans le fichier CSV précédemment ouvert, en utilisant le dialecte 'excel' pour le formatage.
        w = csv.writer(f, dialect='excel')
        #On écrit une ligne d'en-tête dans le fichier CSV avec les noms de colonnes de la table etudiants.
        w.writerow(['id','Image de Profile','Matricule','Nom','Post Nom','Téléphone','Email','Addresse','Date modification'])
        for occurence in donneesBrutes:  #On itère sur chaque ligne de données récupérée et écrit chaque ligne dans le fichier CSV.
            w.writerow(occurence)
    print("sauvegardé: etudiants_"+dateFinal+".csv")
    #On valide toutes les modifications apportées à la base de données depuis la dernière validation.
    conn.commit()
    #On ferme la connexion à la base de données
    conn.close()
    #On affiche une boîte de dialogue d'information vide avec le message "Fichier Excel téléchargé" pour indiquer que l'exportation a réussi.
    messagebox.showinfo("","Fichier Excel téléchargé")
```

Gestion des profiles des étudiants (Cont.)

- Code pour récupérer les données de la BDD à chaque que on appelle cette fonction

```python
#Cette fonction lit toutes les lignes de la table 'etudiants' d'une base de données MySQL et les renvoie sous forme de liste de tuples
#On va commencer à l'appeler pour afficher les données (ex. les données ajouter instantanément, les données supprimées...)
def LireLesDonneesDeLaBDD():
    #On appelle la méthode connection qui se connecte à la BDD définie ci-haut
    conn=connection()
    #On crée un objet curseur à partir de la connexion à la base de données.
    #Le curseur est utilisé pour exécuter des requêtes SQL et récupérer les résultats.
    cursor=conn.cursor()
    #On vérifie si la connexion à la base de données est active en envoyant une requête ping.
    cursor.connection.ping()
    #On définit une requête SQL pour sélectionner toutes les lignes de la table etudiants et les ordonner par id de manière décroissante.
    sql=f"SELECT * FROM etudiants ORDER BY `id` DESC"
    #On exécute la requête SQL pour récupérer les données de la table etudiants.
    cursor.execute(sql)
    #On récupère toutes les lignes de résultats de la requête et les stocke dans la variable resultats sous forme de liste de tuples.
    resultats=cursor.fetchall()
    #On valide toutes les modifications apportées à la base de données depuis la dernière validation.
    #Dans ce cas, il n'y a pas de modification, mais c'est une bonne pratique de valider explicitement les transactions.
    conn.commit()
    #On ferme la connexion
    conn.close()
    #On retourne les résultats
    return resultats
```

Gestion des profiles des étudiants (Cont.)

- Code de la fonction renderTreeView() qui affiche le treeview (datagrid) contenant les données
- La fonction renderTreeView() prend des données en entrée, crée un tableau interactif (ttk.Treeview) dans une interface tkinter, affiche les données dans ce tableau et permet de les faire défiler verticalement à l'aide d'une barre de défilement.

```
#Ce code définit une fonction renderTreeView(data) qui crée et affiche un widget ttk.Treeview dans un cadre (ttk.Frame) sur une fenêtre tkinter
def renderTreeVIew(data):
    #global treeFrame
    #On crée un cadre (Frame) tkinter pour contenir le widget ttk.Treeview, en prenant mainCanvas comme parent.
    #mainCanvas est défini en bas de notre code comme un widget de canevas (Canvas) tkinter qui est créé avec les caractéristiques suivantes :
                #mainWindow est utilisé comme parent pour le canevas. mainWindow est aussi défini en bas de mon code
                #Le fond (bg) du canevas est défini comme blanc (#FFFFFF).
                #La hauteur du canevas est de 720 pixels et la largeur est de 1080 pixels.
                #Les bordures (bd), l'épaisseur du surlignage (highlightthickness) et le relief (relief) sont configurés pour donner
                #une apparence de bordure au canevas.
    treeFrame=ttk.Frame(mainCanvas)
    #On place le cadre à des coordonnées spécifiques sur le parent (mainCanvas) avec une taille spécifiée.
    treeFrame.place(x=270.0,y=130.0,width=760.0,height=535.0)
    #global treeScroll
    #On crée une barre de défilement verticale (Scrollbar) associée au cadre (treeFrame).
    #Cependant, le widget lui-même ne sera pas visible tant qu'il n'aura pas été placé dans la fenêtre principale ou attaché à un autre widget.
    #Pour que la barre de défilement soit visible et fonctionnelle, vous devez l'attacher à un widget qui peut être défilé,
    #comme le ttk.Treeview dans votre cas. On va donc appeller ce scrollbar (défilement) dans treeview (notre datagrid)
    treeScroll=ttk.Scrollbar(treeFrame)
    #On place la barre de défilement à droite du cadre, et on la fait remplir verticalement (fill="y").
    treeScroll.pack(side="right",fill="y")
```

Gestion des profiles des étudiants (Cont.)

- Code de la fonction renderTreeView() qui affiche le datagrid contenant les données (Cont.)

```
    #Déclarer une variable comme globale à l'intérieur d'une fonction signifie que la variable peut être utilisée et modifiée à l'intérieur
    #de cette fonction, mais elle n'est pas accessible à l'extérieur de la fonction.
    #treeview est déclarée comme globale à l'intérieur de la fonction renderTreeView() mais utilisée à l'intérieur d'une boucle for,
    #cela signifie que treeview est traitée comme une variable globale locale à la fonction mais elle est réinitialisée à chaque itération
    #de la boucle for.
    #Cela peut être utile si vous souhaitez réinitialiser ou modifier treeview à chaque itération de la boucle, sans affecter sa portée globale.
    global treeview
    #On définit les noms des colonnes du ttk.Treeview.
    colonnes=("Matricule","Nom","Post Nom","Téléphone","Email","Addresse","Date modification")
    #On crée le ttk.Treeview avec les paramètres suivants :
         ##treeFrame comme parent.
         ##show="headings" pour afficher uniquement les en-têtes des colonnes.
         ##style="mystyle.Treeview" pour appliquer un style personnalisé. Nous avons configuré ce style personnalisé en bas de notre code
         ##yscrollcommand=treeScroll.set pour connecter la barre de défilement verticale au ttk.Treeview.
         ##yscrollcommand=treeScroll.set nous permet de lier la barre de défilement (treeScroll) déclarée ci-haut pour l'attacher au treeview
         ##columns=colonnes pour définir les colonnes.
    treeview=ttk.Treeview(treeFrame,show="headings",style="mystyle.Treeview",yscrollcommand=treeScroll.set,columns=colonnes)
    #On configure les en-têtes de colonne avec treeview.heading()
    #"Matricule" : C'est l'identifiant de la colonne dans le ttk.Treeview. Cet identifiant est utilisé pour référencer la colonne
    #lors de la configuration ou de l'insertion de données dans cette colonne.
    #text="Matricule" : C'est le texte affiché dans l'en-tête de la colonne.
   #Dans cet exemple, le texte "Matricule" sera affiché en tant qu'en-tête de la colonne.
    #anchor="w" : Cela définit l'ancrage du texte dans l'en-tête de la colonne.
    #Dans ce cas, "w" signifie "west" (ouest), ce qui indique que le texte sera aligné à gauche dans l'en-tête de la colonne.
    treeview.heading("Matricule",text="Matricule",anchor="w")
    treeview.heading("Nom",text="Nom",anchor="w")
    treeview.heading("Post Nom",text="Post Nom",anchor="w")
    treeview.heading("Téléphone",text="Téléphone",anchor="w")
    treeview.heading("Email",text="Email",anchor="center")
    treeview.heading("Addresse",text="Addresse",anchor="w")
    treeview.heading("Date modification",text="Date modification",anchor="w")
```

Gestion des profiles des étudiants (Cont.)

- Code de la fonction renderTreeView() qui affiche le datagrid contenant les données (Cont.)

```
#On configure les largeurs de colonne avec treeview.column().
treeview.column("Matricule",width=90)
treeview.column("Nom",width=90)
treeview.column("Post Nom",width=100)
treeview.column("Téléphone",width=90)
treeview.column("Email",width=130)
treeview.column("Addresse",width=98)
treeview.column("Date modification",width=270)
#Boucle pour vider les données du treeview
#Cela peut être utile dans certaines situations où vous souhaitez rafraîchir ou réinitialiser le contenu du ttk.Treeview,
#par exemple avant de le remplir à nouveau avec de nouvelles données.
#treeview.get_children() est une méthode utilisée pour récupérer les identifiants des enfants d'un widget ttk.Treeview.
#Dans ce contexte, les enfants sont les éléments (ou lignes) présents dans le ttk.Treeview.
for data in treeview.get_children():
    #On supprime toutes les lignes actuelles du ttk.Treeview avec treeview.delete(data).
    treeview.delete(data)
#On parcours les données (data) passées à la fonction et les insère dans le ttk.Treeview avec treeview.insert(),
#en spécifiant l'identifiant (iid) de chaque ligne.
for array in data:
    #Dans le code ci-apres:
       #'' : C'est l'identifiant de l'élément parent dans le ttk.Treeview. En spécifiant '', vous indiquez que l'élément est ajouté au niveau
       #le plus haut du ttk.Treeview, ce qui en fait un élément racine.
       #tk.END : C'est l'indice où l'élément sera inséré parmi les enfants de l'élément parent. En utilisant tk.END,
       #vous ajoutez l'élément à la fin de la liste des enfants de l'élément parent, ce qui le place en dernière position.
       #values=array[2:] : C'est une liste des valeurs à afficher dans les colonnes de l'élément. Dans notre cas,
       #array[2:] signifie que les valeurs à partir de l'indice 2 de array seront affichées dans les colonnes du ttk.Treeview.
       #iid=array[0] : C'est l'identifiant unique de l'élément. Il est important que cet identifiant soit unique parmi tous les éléments
       #du ttk.Treeview. Dans notre cas, array[0] est utilisé comme identifiant,
       #ce qui signifie que la première valeur de array est utilisée comme identifiant de l'élément.
    treeview.insert('',tk.END,values=array[2:],iid=array[0])
    print(array)
#On place le ttk.Treeview à l'intérieur du cadre avec des dimensions spécifiées.
#x=0, y=0 : Cela place le coin supérieur gauche du ttk.Treeview aux coordonnées (0, 0) de treeFrame,
#ce qui signifie que le ttk.Treeview sera positionné dans le coin supérieur gauche de treeFrame.
#width=745.0, height=535.0 : Cela définit la largeur du ttk.Treeview à 745 pixels et la hauteur à 535 pixels,
#ce qui détermine la taille totale du ttk.Treeview.
#On a donc placé et dimensionné le ttk.Treeview dans treeFrame de manière à occuper toute la zone spécifiée par les coordonnées et les dimensions.
treeview.place(x=0,y=0,width=745.0,height=535.0)
#On configure la barre de défilement verticale pour contrôler le défilement du ttk.Treeview avec treeScroll.config(command=treeview.yview).
#treeScroll : C'est la barre de défilement verticale que vous avez créée pour le ttk.Treeview.
#config() : C'est une méthode pour configurer les options d'un widget tkinter.
#command=treeview.yview : Cela configure la barre de défilement pour appeler la méthode yview() du ttk.Treeview
#Lorsque la barre de défilement est déplacée verticalement. La méthode yview() permet au ttk.Treeview de se déplacer verticalement
#pour afficher différentes parties de son contenu lorsque l'utilisateur fait défiler la barre de défilement.
treeScroll.config(command=treeview.yview)
```

Gestion des profiles des étudiants (Cont.)

- Ces fonctions sont utilisées pour gérer les chemins d'accès aux images qui constituent le design et bouton de notre application.

```
#Cette fonction prend une fenêtre tkinter (window) en argument et détruit cette fenêtre en appelant window.destroy().
#Permet de sauvegarder temporairement l'image du profile d'un étudiant ajouter dans la BDD
def closeWindow(window):
    window.destroy()
    #Elle vérifie si un fichier spécifique existe ("./assets/uploaded/temp.png") en utilisant os.path.exists().
    if os.path.exists("./assets/uploaded/temp.png"):
        #Si le fichier existe, elle le supprime en appelant os.remove().
        os.remove("./assets/uploaded/temp.png")
#Cette fonction prend une chaîne de caractères (str) en argument et retourne un chemin d'accès relatif à un fichier dans le répertoire
#./assets/frame1/. N.B: assets/frame1/ contient les images (Clear, submit, cancel...) qui sont utilisées comme button lors de modification,
#insertion, etc. des données dans notre application. Ca permet de les charger lors du clic sur le bouton Ajouter du menu principal
def addWindowAssets(str):
    return f"./assets/frame1/{str}"
#Semblable à addWindowAssets(), cette fonction prend une chaîne de caractères en argument et retourne un chemin d'accès relatif
#à un fichier dans le répertoire. Elle charge ces images des boutton lorsque l'on a ckiqué sur modifier
def editWindowAssets(str):
    return f"./assets/frame1/{str}"
#Meme chose. Mais lorsque l'on a cliqué sur le boutton Afficher du menu principal
def viewWindowAssets(str):
    return f"./assets/frame1/{str}"
#Cette méthode charge les images se trouvant dans le dossier frame0 qui contient les boutons Ajouter, Supprimer, Modifier, ...
#Elle les charges dans le menu principal
def mainWindowAssets(str):
    return f"./assets/frame0/{str}"
```

Gestion des profiles des étudiants (Cont.)

- Cette fonction permet de supprimer un profile d'étudiant dans la BDD

```
#Ce code définit une fonction supprimerEtudiant() qui permet de supprimer un étudiant de la base de données en fonction
#de la ligne sélectionnée dans le ttk.Treeview
def supprimerEtudiant():
    try:
        #Cette ligne récupère l'identifiant (matricule) de l'étudiant à supprimer en accédant aux valeurs de la première colonne
        #de la ligne sélectionnée dans le ttk.Treeview.
        donneeSupprimer = str(treeview.item(treeview.selection()[0])['values'][0])
        #On gère les exceptions qui peuvent survenir lors de la récupération de l'identifiant. Si aucune ligne n'est sélectionnée,
        #une boîte de dialogue d'avertissement est affichée.
    except:
        messagebox.showwarning("Information", "Veuiller séléctionner une ligne de données")
        return
    #Une boîte de dialogue demande à l'utilisateur s'il souhaite vraiment supprimer l'étudiant sélectionné.
    #Si la réponse est "non", la fonction se termine.
    decision = messagebox.askquestion("Attention", "Voulez-vous supprimer l'étudiant séléctionné?")
    if decision != "yes":
        return
    #Ce else se charge de supprimer l'étudiant de la base de données en utilisant une requête SQL DELETE. Si une erreur se produit
    #lors de la suppression, une boîte de dialogue d'information est affichée.
    else:
        try:
            conn = connection()
            cursor = conn.cursor()
            cursor.execute(f"DELETE FROM etudiants WHERE matricule='{str(donneeSupprimer)}'")
            conn.commit()
            conn.close()
        except:
            #En cas d'erreur de connexion, on affiche une boîte de dialogue
            messagebox.showinfo("Erreur", "Désolé, une erreur s'est produite")
            return
    print(donneeSupprimer)
    #Après la suppression de l'étudiant, le ttk.Treeview est mis à jour en appelant renderTreeView() avec les données mises à jour obtenues
    #en appelant LireLesDonneesDeLaBDD() que nous avons déclaré ci-haut pour raffraichir les actions dans la BDD.
    renderTreeVIew(LireLesDonneesDeLaBDD())
```

Gestion des profiles des étudiants (Cont.)

- Cette fonction sera chaque fois appelée pour modifier un profile d'étudiant dans la BDD

```
#Ce code définit une fonction modifierEtudiant() qui récupère les données d'un étudiant sélectionné dans le ttk.Treeview afin de les utiliser
#pour une éventuelle modification.
def modifierEtudiant():
    # Cette ligne initialise une liste selectionDonneeEtudiant avec des valeurs (Matricule, Nom, ...) par défaut.
    selectionDonneeEtudiant = [0,0,0,0,0,0]
    try:
        #On itère sur les six premières valeurs de la ligne sélectionnée dans le ttk.Treeview et les stocke dans selectionDonneeEtudiant.
        for i in range(0,6):
            selectionDonneeEtudiant[i] = str(treeview.item(treeview.selection()[0])['values'][i])
        #On gère les exceptions qui peuvent survenir lors de la récupération des données.
        #Si aucune ligne n'est sélectionnée, une boîte de dialogue d'avertissement est affichée
    except:
        messagebox.showwarning("Information", "Veuiller séléctionner une ligne de données")
        return
    # Cette ligne affiche les données de l'étudiant sélectionné dans la console. Cela peut être utile pour le débogage.
    print(selectionDonneeEtudiant)
    #Cette ligne appelle la fonction renderEditWindow() qui sera définie en bas en lui passant les données de l'étudiant sélectionné.
    #Cette fonction est utilisée pour afficher une fenêtre de modification où les données de l'étudiant peuvent être modifiées.
    renderEditWindow(selectionDonneeEtudiant)
```

Gestion des profiles des étudiants (Cont.)

- Cette fonction sera chaque fois appelée pour afficher un profile d'étudiant dans la BDD

```
#Ce code définit une fonction voirEtudiant() qui récupère les données d'un étudiant sélectionné dans le ttk.Treeview afin de les utiliser
#pour une éventuelle vue des données d'un étudiant.
def voirEtudiant():
    # Cette ligne initialise une liste selectionDonneeEtudiant avec des valeurs (Matricule, Nom, ...) par défaut.
    selectionDonneeEtudiant = [0,0,0,0,0,0]
    try:
        #On itère sur les six premières valeurs de la ligne sélectionnée dans le ttk.Treeview et les stocke dans selectionDonneeEtudiant.
        for i in range(0,6):
            selectionDonneeEtudiant[i] = str(treeview.item(treeview.selection()[0])['values'][i])
        ##On gère les exceptions qui peuvent survenir lors de la récupération des données.
        #Si aucune ligne n'est sélectionnée, une boîte de dialogue d'avertissement est affichée
    except:
        messagebox.showwarning("Information", "Veuiller séléctionner une ligne de données")
        return
    # Cette ligne affiche les données de l'étudiant sélectionné dans la console. Cela peut être utile pour le débogage.
    print(selectionDonneeEtudiant)
    #Cette ligne appelle la fonction renderViewWindow() qui sera définie en bas en lui passant les données de l'étudiant sélectionné.
    #Cette fonction est utilisée pour afficher une fenêtre d'affichage où les données de l'étudiant peuvent être vues.
    renderViewWindow(selectionDonneeEtudiant)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**.

```
#La fonction ici retourne le menu d'ajout de l'étudiant
def renderAddWindow():
    #Cette fonction est appelée lorsque l'utilisateur clique sur le bouton "Ajouter". Elle récupère les données saisies par l'utilisateur,
    #vérifie si elles sont complètes, vérifie si le matricule de l'étudiant est déjà utilisé, puis ajoute l'étudiant à la base de données.
    def ajoutEtudiant():
        #Cette ligne génère un nom aléatoire pour l'image de profil de l'étudiant en utilisant la  generer_caracteres_aleatoires() et l'ajoute avec
        #l'extension ".png". La fonction generer_caracteres_aleatoires() a été déclarée dans les slides précédents
        nomDeImageProfile = f"{generer_caracteres_aleatoires()}.png"
        #On récupère le matricule de l'étudiant à partir du champ de saisie ajoutInputMatricule qui sera défini en bas du code. Et on le fait pour
        #toutes les variables
        matricule= str(ajoutInputMatricule.get())
        nom = str(ajoutInputNom.get())
        postnom = str(ajoutInputPostnom.get())
        telephone = str(ajoutInputTelephone.get())
        email = str(ajoutInputEmail.get())
        addresse = str(ajoutInputAddresse.get())
        #On vérifie si l'un des champs est vide. Si c'est le cas, affiche un message d'erreur et arrête la fonction.
        if (matricule == "" or matricule == " ") or (nom == "" or nom == " ") or (postnom == "" or postnom == " ") or
        (telephone == "" or telephone == " ") or (email == "" or email == " ") or (addresse == "" or addresse == " "):
            #addCanvas fait référence à un objet Canvas qui est utilisé comme parent pour la boîte de dialogue messagebox.showinfo
            #Nous allons déclaré en bas
            messagebox.showinfo("Erreur", "Veuillez compléter l'entrée vide", parent=addCanvas)
            return
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)

```
#Sinon, tente de se connecter à la base de données et vérifie si le matricule de l'étudiant existe déjà.
#Si c'est le cas, affiche un message d'erreur et arrête la fonction.
else:
    try:
        conn = connection()
        cursor = conn.cursor()
        cursor.execute(f"SELECT * FROM etudiants WHERE matricule = '{matricule}'")
        if cursor.fetchone() is not None:
            messagebox.showinfo("Erreur", "Ce Matricule existe déjà", parent=addCanvas)
            conn.close()
            return
        global profile_img
        #On ouvre l'image temporaire sélectionnée par l'utilisateur, la redimensionne, la convertit en mode RGB et
        #la sauvegarde avec le nom généré aléatoirement dans le dossier des images uploadées.
        profile_img = Image.open("./assets/uploaded/temp.png")
        #Cette ligne de code redimensionne l'image profile_img à une taille de 145x145 pixels en utilisant
        #l'algorithme de rééchantillonnage Lanczos.
        #L'algorithme de rééchantillonnage Lanczos est souvent utilisé pour réduire les pertes de qualité lors du redimensionnement des images,
        #en particulier pour les réductions de taille importantes.
        profile_img = profile_img.resize((145, 145), resample=Image.LANCZOS)
        profile_img = profile_img.convert("RGB")
        profile_img.save(f"./assets/uploaded/{nomDeImageProfile}", format="PNG")
        if os.path.exists("./assets/uploaded/temp.png"):
            #On supprime l'image temporaire sélectionnée par l'utilisateur.
            os.remove("./assets/uploaded/temp.png")
        #On exécute une requête SQL pour insérer les données de l'étudiant (y compris le nom de l'image de profil généré aléatoirement)
        #dans la table des étudiants de la base de données.
        cursor.execute(f"INSERT INTO etudiants (image_profile, matricule, nom, postnom, phone, email, addresse, date_mise_a_jour) VALUES
        ('{nomDeImageProfile}', '{matricule}', '{nom}', '{postnom}', '{telephone}', '{email}', '{addresse}', '{getDateActuelle()}')")
        conn.commit()
        #On ferme la connexion à la base de données.
        conn.close()
    except Exception as e:
        print(e)
        #Si une exception se produit lors de l'exécution des étapes précédentes, affiche un message d'erreur.
        messagebox.showinfo("Erreur", "Une erreur s'est produite lors de l'ajout de l'étudiant", parent=addCanvas)
        return
#On ferme la fenêtre d'ajout d'étudiant. La fonction closeWindow() a été déclarée en haut et permet de supprimer les images temporaires
closeWindow(addWindow)
#On met à jour l'affichage de la liste des étudiants avec les données actualisées depuis la base de données.
renderTreeVIew(LireLesDonneesDeLaBDD())
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)
- N.B: Ce code qui suit se trouve dans la fonction renderAddWindow() au même niveau que la sous fonction def AjoutEtudiant()

```
#Cette fonction setPreviewPic(filepath) est utilisée pour charger et afficher une image à partir d'un fichier spécifié filepath dans la
#fenêtre addWindow à l'aide de la bibliothèque tkinter.
    def setPreviewPic(filepath):
        #On déclare que la variable image est globale, ce qui signifie qu'elle peut être utilisée et modifiée à l'extérieur de la fonction.
        global image
        #On utilise un bloc try-except pour gérer les erreurs potentielles lors du chargement de l'image.
        #Si une erreur se produit, elle est imprimée à la console.
        try:
            #On charge l'image à partir du fichier spécifié filepath dans la variable image en tant qu'objet PhotoImage de tkinter.
            #Cet objet PhotoImage est utilisé pour représenter l'image dans l'interface utilisateur.
            image = PhotoImage(master=addWindow, file=filepath)
            #On crée une image dans le canevas addCanvas à la position (112.0, 168.0) en utilisant l'objet PhotoImage chargé précédemment.
            #Cela affiche l'image dans la fenêtre addWindow à l'emplacement spécifié.
            addCanvas.create_image(112.0, 168.0, image=image)
        except Exception as e:
            print(e)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)
- N.B: Ce code qui suit se trouve dans la fonction renderAddWindow() au même niveau que la sous fonction def AjoutEtudiant()

```
#Cette fonction permet à l'utilisateur de sélectionner un fichier image à partir d'une boîte de dialogue de sélection de fichier.
    def selectPic():
        global filepath
        #On ouvre une boîte de dialogue de sélection de fichier pour permettre à l'utilisateur de choisir un fichier image.
        #Les options spécifiées incluent le master=addCanvas pour attacher la boîte de dialogue à la fenêtre addCanvas,
        #initialdir=os.getcwd() pour définir le répertoire initial sur le répertoire de travail actuel,
        #title="Sélection de l'image" pour définir le titre de la boîte de dialogue et
        #filetypes=[("fichiers des images","*.png *.jpg *.jpeg"),] pour spécifier les types de fichiers autorisés.
        filepath = filedialog.askopenfilename(
            master=addCanvas,
            initialdir=os.getcwd(),
            title="Sélection de l'image",
            filetypes=[("fichiers des images","*.png *.jpg *.jpeg"),]
        )
        global profile_img
        #On charge l'image sélectionnée à partir du chemin filepath dans la variable profile_img en tant qu'objet Image de la bibliothèque Pillow.
        profile_img = Image.open(filepath)
        #On redimensionne l'image chargée à une taille de 145x145 pixels en utilisant l'algorithme Lanczos de rééchantillonnage,
        #qui est généralement utilisé pour obtenir une meilleure qualité lors du redimensionnement des images.
        profile_img = profile_img.resize((145, 145), resample=Image.LANCZOS)
        #On convertit l'image en mode RGB si elle n'est pas déjà dans ce mode.
        #Cela garantit que l'image est compatible avec la sauvegarde ultérieure au format PNG.
        profile_img = profile_img.convert("RGB")
        #On enregistre l'image redimensionnée et convertie au format PNG dans le répertoire ./assets/uploaded/ avec le nom temp.png.
        profile_img.save(f"./assets/uploaded/temp.png", format="PNG")
        #On appelle la fonction setPreviewPic pour afficher l'image sélectionnée dans l'interface utilisateur.
        setPreviewPic(f"./assets/uploaded/temp.png")
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)
- N.B: toujours au même niveau que la sous fonction def AjoutEtudiant(). Ce code permet donc d'afficher le menu pour ajouter un profile étudiant

```
#Ce code (tjrs dans la fonction renderAddWindow()) crée une fenêtre pour ajouter un étudiant dans un système de gestion des profils étudiants.
    print("Rendre la fenêtre d'ajout des Etudiants")
    #On crée une nouvelle fenêtre principale en utilisant la classe Tk de Tkinter.
    addWindow = Tk()
    #On définit le titre de la fenêtre.
    addWindow.title('Ajouter un Etudiant - Système de gestion des profiles des étudiants')
    #On définit la taille initiale de la fenêtre.
    addWindow.geometry("720x480")
    #On configure les paramètres de la fenêtre, comme la couleur de fond.
    addWindow.configure(bg = "#FFFFFF")
    #On crée un canevas à l'intérieur de la fenêtre pour afficher des éléments graphiques.
    #ce candvas est appelé en haut dans le sous fonction def ajouterEtudiant()
    #relief='ridge' est un paramètre utilisé pour spécifier le style de relief d'un widget dans Tkinter.
    #Le relief définit l'apparence en relief ou en creux d'un widget, ce qui peut aider à donner une impression de profondeur et de texture.
    #Dans ce cas, relief='ridge' donne à un canevas (Canvas) un relief en forme de crête, ce qui signifie que les bords du canevas semblent
    #saillants par rapport au contenu du canevas. Cela peut être utile pour donner un aspect visuel distinctif à un élément de
    #l'interface utilisateur. D'autres valeurs possibles pour le paramètre relief incluent 'flat', 'sunken', 'raised', 'groove', et 'solid',
    #chacune offrant un style de relief différent.
    addCanvas = Canvas(addWindow,bg = "#FFFFFF",height = 480,width = 720,bd = 0,highlightthickness = 0,relief = "ridge")
    #On place le canevas dans la fenêtre à une position spécifiée.
    addCanvas.place(x = 0, y = 0)
    #On charge des images à afficher dans la fenêtre (N.B: image_1.png c'est l'image grise de l'interface de notre application).
    #On apppelle la méthode addWindowAssets() qui charge les images se trouvant dans assets/frame1. Les images 1, 2, 3 vont se charger
    #On crée ensuite des images à afficher sur le canevas pour ces trois images chargées avec addCanvas.create_image().
    image_image_1 = PhotoImage(master=addWindow, file=addWindowAssets("image_1.png"))
    addCanvas.create_image(360.0, 264.0, image=image_image_1)
    image_image_2 = PhotoImage(master=addWindow, file=addWindowAssets("image_2.png"))
    addCanvas.create_image(360.0, 24.0, image=image_image_2)
    addCanvas.create_text(49.0, 10.0, anchor="nw", text="Ajouter Etudiant", fill="#FFFFFF", font=("Inter SemiBold", 24 * -1))
    image_image_3 = PhotoImage(master=addWindow, file=addWindowAssets("image_3.png"))
    addCanvas.create_image(28.0, 24.0, image=image_image_3)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)
- N.B: toujours au même niveau que la sous fonction def AjoutEtudiant(). Ce code permet donc d'afficher le menu pour ajouter un profile étudiant (Cont.)

```
#Entry(...): crée ici des zones de texte pour permettre à l'utilisateur de saisir des informations du Matricule à l'adresse.
#addCanvas.create_text(...): crée des zones de texte pour étiqueter les champs de saisie texte input seront des images
#Pour la zone de texte Matricule par exemple,
    #Les images image_5 et image_2 sont utilisées pour décorer et étiqueter visuellement la zone de saisie du matricule,
    #en ajoutant des éléments graphiques pour améliorer l'interface utilisateur.
    #Elles servent principalement à des fins de design et d'esthétique, en rendant l'interface plus conviviale et attrayante.
#N.B: le paramètre master indique le widget parent auquel la PhotoImage est associée.
#Cela signifie que l'image sera affichée dans le widget spécifié comme parent, qui dans ce cas est addWindow.
#Ainsi, lorsque vous créez une PhotoImage avec addWindow comme maître,
#cette image sera affichée dans addWindow ou dans un widget enfant de addWindow, si spécifié.
#ajoutInputMatricule, ajoutInputNom, etc. sont récuperées comme valeurs en haut dans la sous fonction ajoutEtudiant()
    #input de matricule
    image_image_5 = PhotoImage(master=addWindow, file=addWindowAssets("image_5.png"))
    addCanvas.create_image(456.0, 104.0, image=image_image_5)
    entry_image_2 = PhotoImage(master=addWindow, file=addWindowAssets("entry_2.png"))
    addCanvas.create_image(455.0, 109.5, image=entry_image_2)
    ajoutInputMatricule = Entry(master=addWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    ajoutInputMatricule.place(x=325.0, y=98.0, width=260.0, height=21.0)
    addCanvas.create_text(325.0, 85.0, anchor="nw", text="Matricule", fill="#000000", font=("Inter", 11 * -1))
    #input du nom
    image_image_4 = PhotoImage(master=addWindow, file=addWindowAssets("image_4.png"))
    addCanvas.create_image(457.0, 166.0, image=image_image_4)
    entry_image_1 = PhotoImage(master=addWindow, file=addWindowAssets("entry_1.png"))
    addCanvas.create_image(456.0, 171.5, image=entry_image_1)
    ajoutInputNom = Entry(master=addWindow,bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    ajoutInputNom.place(x=326.0, y=160.0, width=260.0, height=21.0)
    addCanvas.create_text(326.0, 147.0, anchor="nw", text="Nom", fill="#000000", font=("Inter", 11 * -1))
    #input du postnom
    image_image_6 = PhotoImage(master=addWindow, file=addWindowAssets("image_6.png"))
    addCanvas.create_image(457.0,230.0,image=image_image_6)
    entry_image_3 = PhotoImage(master=addWindow, file=addWindowAssets("entry_3.png"))
    addCanvas.create_image(456.0,235.5,image=entry_image_3)
    ajoutInputPostnom = Entry(master=addWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    ajoutInputPostnom.place(x=326.0,y=224.0,width=260.0,height=21.0)
    addCanvas.create_text(326.0,211.0,anchor="nw",text="Post Nom",fill="#000000",font=("Inter", 11 * -1))
```

Gestion des profiles des étudiants (Cont.)

```
    #input du téléphone
    image_image_7 = PhotoImage(master=addWindow, file=addWindowAssets("image_7.png"))
    addCanvas.create_image(166.0,294.0,image=image_image_7)
    entry_image_4 = PhotoImage(master=addWindow, file=addWindowAssets("entry_4.png"))
    addCanvas.create_image(165.0,299.5,image=entry_image_4)
    ajoutInputTelephone = Entry(master=addWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    ajoutInputTelephone.place(x=35.0,y=288.0,width=260.0,height=21.0)
    addCanvas.create_text(35.0,275.0,anchor="nw",text="Téléphone",fill="#000000",font=("Inter", 11 * -1))
    #input de l'email
    image_image_8 = PhotoImage(master=addWindow, file=addWindowAssets("image_8.png"))
    addCanvas.create_image(457.0,294.0,image=image_image_8)
    entry_image_5 = PhotoImage(master=addWindow, file=addWindowAssets("entry_5.png"))
    addCanvas.create_image(456.0,299.5,image=entry_image_5)
    ajoutInputEmail = Entry(master=addWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    ajoutInputEmail.place(x=326.0,y=288.0,width=260.0,height=21.0)
    addCanvas.create_text(326.0,275.0,anchor="nw",text="Email",fill="#000000",font=("Inter", 11 * -1))
    #input de l'addresse
    image_image_9 = PhotoImage(master=addWindow, file=addWindowAssets("image_9.png"))
    addCanvas.create_image(311.0, 358.0, image=image_image_9)
    entry_image_6 = PhotoImage(master=addWindow, file=addWindowAssets("entry_6.png"))
    addCanvas.create_image(310.5, 363.5, image=entry_image_6)
    ajoutInputAddresse = Entry(master=addWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    ajoutInputAddresse.place(x=35.0, y=352.0, width=551.0, height=21.0)
    addCanvas.create_text(35.0,339.0,anchor="nw",text="Addresse",fill="#000000",font=("Inter", 11 * -1))
    image_image_10 = PhotoImage(master=addWindow, file=addWindowAssets("image_10.png"))
    addCanvas.create_image(166.0, 167.0, image=image_image_10)
    #Cette ligne appelle la fonction setPreviewPic avec l'argument "./assets/uploaded/default.png".
    #Cette fonction est utilisée pour afficher une image de prévisualisation dans la fenêtre.
    #Elle charge l'image à partir du chemin spécifié et l'affiche dans le canevas (addCanvas) à un emplacement spécifique.
    setPreviewPic("./assets/uploaded/default.png")
#On crée des boutons pour effectuer des actions, comme sélectionner une image ou soumettre un formulaire.
#On place le bouton de soumission dans la fenêtre à une position spécifiée.
    button_image_2 = PhotoImage(master=addWindow, file=addWindowAssets("button_2.png"))
    selectImageBtn = Button(master=addWindow, image=button_image_2, borderwidth=0, highlightthickness=0, command=selectPic, relief="flat")
    selectImageBtn.place(x=196.0, y=215.0, width=96.0, height=25.0)
    button_image_3 = PhotoImage(master=addWindow, file=addWindowAssets("button_3.png"))
    addSubmitBtn = Button(master=addWindow, image=button_image_3, borderwidth=0, highlightthickness=0, command=ajoutEtudiant, relief="flat")
    addSubmitBtn.place(x=28.0, y=402.0, width=96.0, height=25.0)
    button_image_4 = PhotoImage(master=addWindow, file=addWindowAssets("button_4.png"))
    cancelBtn = Button(master=addWindow, image=button_image_4, borderwidth=0, highlightthickness=0, command=lambda: closeWindow(addWindow),
                       relief="flat")
    cancelBtn.place(x=137.0, y=402.0, width=96.0, height=25.0)
    addWindow.resizable(False, False) #On configure si la fenêtre peut être redimensionnée par l'utilisateur.
#On lance la boucle principale de la fenêtre pour afficher les éléments et attendre les interactions de l'utilisateur.
    addWindow.mainloop()
```

- Ce code définit une fonction renderAddWindow() qui affiche une fenêtre **d'ajout d'étudiants**(Cont.)
- N.B: toujours au même niveau que la sous fonction def AjoutEtudiant(). Ce code permet donc d'afficher le menu pour ajouter un profile étudiant (Cont.)

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderEditWindow() qui affiche une fenêtre **de modification d'un étudiant.** Ca fonctionne presque comme **renderAddWindow()**

```
def renderEditWindow(selectionDonneeEtudiant):
    def modifierEtudiant():
        nomDeImageProfile = f"{generer_caracteres_aleatoires()}.png"
        idEtudiantSelectionne = selectionDonneeEtudiant[0]
        matricule = str(modifieMatriculeEtudiantInput.get())
        nom = str(modifieNomEtudiantInput.get())
        postnom = str(modifiePostnomEtudiantInput.get())
        telephone= str(modifieTelephoneEtudiantInput.get())
        email = str(modifieEmailEtudiantInput.get())
        addresse = str(modifieAddresseEtudiantInput.get())
        if (matricule == "" or matricule == " ") or (nom == "" or nom == " ") or (postnom == "" or postnom == " ") or
        (telephone == "" or telephone == " ") or (email == "" or email == " ") or (addresse == "" or addresse == " "):
            messagebox.showinfo("Erreur", "Veuillez compléter l'entrée vide",parent=editCanvas)
            return
        else:
            try:
                try:
                    global profile_img
                    profile_img = Image.open("./assets/uploaded/temp.png")
                    profile_img = profile_img.resize((145, 145), resample=Image.LANCZOS)
                    profile_img = profile_img.convert("RGB")
                    profile_img.save(f"./assets/uploaded/{nomDeImageProfile}", format="PNG")
                    conn = connection()
                    cursor = conn.cursor()
                    cursor.execute(f"SELECT * FROM etudiants WHERE matricule='{matricule}' ")
                    resultat = cursor.fetchone()
                    conn.commit()
                    conn.close()
                    if os.path.exists(f"./assets/uploaded/{resultat[1]}"):
                        os.remove(f"./assets/uploaded/{resultat[1]}")
                    conn = connection()
                    cursor = conn.cursor()
                    cursor.execute(f"UPDATE etudiants SET image_profile='{nomDeImageProfile}',matricule='{matricule}',nom='{nom}',
                    postnom='{postnom}',phone='{telephone}',email='{email}',addresse='{addresse}',date_mise_a_jour='{getDateActuelle()}'
                    WHERE matricule='{idEtudiantSelectionne}' ")
                    conn.commit()
                    conn.close()
                except:
                    conn = connection()
                    cursor = conn.cursor()
                    cursor.execute(f"UPDATE etudiants SET matricule='{matricule}',nom='{nom}',postnom='{postnom}',
                    phone='{telephone}',email='{email}',addresse='{addresse}',date_mise_a_jour='{getDateActuelle()}'
                    WHERE matricule='{idEtudiantSelectionne}' ")
                    conn.commit()
                    conn.close()
                if os.path.exists("./assets/uploaded/temp.png"):
                    os.remove("./assets/uploaded/temp.png")
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderEditWindow() qui affiche une fenêtre **de modification d'un étudiant.** Ca fonctionne presque comme **renderAddWindow() (Cont.)**

```
            except Exception as e:
                print(e)
                messagebox.showinfo("Erreur", "Une erreur est survenue",parent=editCanvas)
                return
        closeWindow(editWindow)
        renderTreeVIew(LireLesDonneesDeLaBDD())

    def setPreviewPic(filepath):
        global image
        try:
            image = PhotoImage(master=editWindow, file=filepath)
            editCanvas.create_image(112.0, 168.0, image=image)
        except Exception as e:
            print(e)

    def selectPic():
        global filepath
        filepath = filedialog.askopenfilename(
            master=editCanvas,
            initialdir=os.getcwd(),
            title="Selection de l'image",
            filetypes=[("Fichiers des images","*.png *.jpg *.jpeg"),]
        )
        global profile_img
        profile_img = Image.open(filepath)
        profile_img = profile_img.resize((145, 145), resample=Image.LANCZOS)
        profile_img = profile_img.convert("RGB")
        profile_img.save(f"./assets/uploaded/temp.png", format="PNG")
        setPreviewPic(f"./assets/uploaded/temp.png")
#On recupère les données venant de treeview apres séléction d'une ligne
    matricule=selectionDonneeEtudiant[0]
    nom=selectionDonneeEtudiant[1]
    postnom=selectionDonneeEtudiant[2]
    telephone=selectionDonneeEtudiant[3]
    email=selectionDonneeEtudiant[4]
    addresse=selectionDonneeEtudiant[5]
    conn = connection()
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM etudiants WHERE matricule='{matricule}' ")
    resultat = cursor.fetchone()
    conn.commit()
    conn.close()
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderEditWindow() qui affiche une fenêtre **de modification d'un étudiant.** Ca fonctionne presque comme **renderAddWindow() (Cont.)**

```
#On crée une fenêtre pour créer un menu de saisi des données à modifier après clic sur le bouton modifier
    editWindow = Tk()
    editWindow.title('Fenêtre de modification - Système de gestion des profils des étudiants')
    editWindow.geometry("720x480")
    editWindow.configure(bg = "#FFFFFF")
    editCanvas = Canvas(editWindow,bg = "#FFFFFF",height = 480,width = 720,bd = 0,highlightthickness = 0,relief = "ridge")
    editCanvas.place(x = 0, y = 0)
    image_image_1 = PhotoImage(master=editWindow, file=editWindowAssets("image_1.png"))
    editCanvas.create_image(360.0, 264.0, image=image_image_1)
    image_image_2 = PhotoImage(master=editWindow, file=editWindowAssets("image_2.png"))
    editCanvas.create_image(360.0, 24.0, image=image_image_2)
    editCanvas.create_text(49.0, 10.0, anchor="nw", text="Modifier Etudiant", fill="#FFFFFF", font=("Inter SemiBold", 24 * -1))
    image_image_3 = PhotoImage(master=editWindow, file=editWindowAssets("image_3.png"))
    editCanvas.create_image(28.0, 24.0, image=image_image_3)
    #input modifier matricule
    image_image_5 = PhotoImage(master=editWindow, file=editWindowAssets("image_5.png"))
    editCanvas.create_image(456.0, 104.0, image=image_image_5)
    entry_image_2 = PhotoImage(master=editWindow, file=editWindowAssets("entry_2.png"))
    editCanvas.create_image(455.0, 109.5, image=entry_image_2)
    modifieMatriculeEtudiantInput = Entry(master=editWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    modifieMatriculeEtudiantInput.place(x=325.0, y=98.0, width=260.0, height=21.0)
    editCanvas.create_text(325.0, 85.0, anchor="nw", text="Matricule", fill="#000000", font=("Inter", 11 * -1))
    modifieMatriculeEtudiantInput.insert(0,matricule)
    #input de modification du nom
    image_image_4 = PhotoImage(master=editWindow, file=editWindowAssets("image_4.png"))
    editCanvas.create_image(457.0, 166.0, image=image_image_4)
    entry_image_1 = PhotoImage(master=editWindow, file=editWindowAssets("entry_1.png"))
    editCanvas.create_image(456.0, 171.5, image=entry_image_1)
    modifieNomEtudiantInput = Entry(master=editWindow,bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    modifieNomEtudiantInput.place(x=326.0, y=160.0, width=260.0, height=21.0)
    editCanvas.create_text(326.0, 147.0, anchor="nw", text="Nom", fill="#000000", font=("Inter", 11 * -1))
    modifieNomEtudiantInput.insert(0,nom)
    #input de modification de post nom
    image_image_6 = PhotoImage(master=editWindow, file=editWindowAssets("image_6.png"))
    editCanvas.create_image(457.0,230.0,image=image_image_6)
    entry_image_3 = PhotoImage(master=editWindow, file=editWindowAssets("entry_3.png"))
    editCanvas.create_image(456.0,235.5,image=entry_image_3)
    modifiePostnomEtudiantInput = Entry(master=editWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    modifiePostnomEtudiantInput.place(x=326.0,y=224.0,width=260.0,height=21.0)
    editCanvas.create_text(326.0,211.0,anchor="nw",text="Post Nom",fill="#000000",font=("Inter", 11 * -1))
    modifiePostnomEtudiantInput.insert(0,postnom)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderEditWindow() qui affiche une fenêtre **de modification d'un étudiant.** Ca fonctionne presque comme **renderAddWindow() (Cont.)**

```
    #input de modification de telephone
    image_image_7 = PhotoImage(master=editWindow, file=editWindowAssets("image_7.png"))
    editCanvas.create_image(166.0,294.0,image=image_image_7)
    entry_image_4 = PhotoImage(master=editWindow, file=editWindowAssets("entry_4.png"))
    editCanvas.create_image(165.0,299.5,image=entry_image_4)
    modifieTelephoneEtudiantInput = Entry(master=editWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    modifieTelephoneEtudiantInput.place(x=35.0,y=288.0,width=260.0,height=21.0)
    editCanvas.create_text(35.0,275.0,anchor="nw",text="Téléphone",fill="#000000",font=("Inter", 11 * -1))
    modifieTelephoneEtudiantInput.insert(0,telephone)
    #input de modification de email
    image_image_8 = PhotoImage(master=editWindow, file=editWindowAssets("image_8.png"))
    editCanvas.create_image(457.0,294.0,image=image_image_8)
    entry_image_5 = PhotoImage(master=editWindow, file=editWindowAssets("entry_5.png"))
    editCanvas.create_image(456.0,299.5,image=entry_image_5)
    modifieEmailEtudiantInput= Entry(master=editWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    modifieEmailEtudiantInput.place(x=326.0,y=288.0,width=260.0,height=21.0)
    editCanvas.create_text(326.0,275.0,anchor="nw",text="Email",fill="#000000",font=("Inter", 11 * -1))
    modifieEmailEtudiantInput.insert(0,email)
    #input modification addresse
    image_image_9 = PhotoImage(master=editWindow, file=editWindowAssets("image_9.png"))
    editCanvas.create_image(311.0, 358.0, image=image_image_9)
    entry_image_6 = PhotoImage(master=editWindow, file=editWindowAssets("entry_6.png"))
    editCanvas.create_image(310.5, 363.5, image=entry_image_6)
    modifieAddresseEtudiantInput = Entry(master=editWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    modifieAddresseEtudiantInput.place(x=35.0, y=352.0, width=551.0, height=21.0)
    editCanvas.create_text(35.0,339.0,anchor="nw",text="editress",fill="#000000",font=("Inter", 11 * -1))
    modifieAddresseEtudiantInput.insert(0,addresse)

    image_image_10 = PhotoImage(master=editWindow, file=editWindowAssets("image_10.png"))
    editCanvas.create_image(166.0, 167.0, image=image_image_10)
#On charge l'image à partir du dossier uploaded dont le nom correspond avec celui de l'image venant de la BDD
    setPreviewPic(f"./assets/uploaded/{resultat[1]}")
#On met les boutons
    button_image_2 = PhotoImage(master=editWindow, file=editWindowAssets("button_2.png"))
    selectImageBtn = Button(master=editWindow, image=button_image_2, borderwidth=0, highlightthickness=0, command=selectPic, relief="flat")
    selectImageBtn.place(x=196.0, y=215.0, width=96.0, height=25.0)
    button_image_3 = PhotoImage(master=editWindow, file=editWindowAssets("button_3.png"))
    submitBtn = Button(master=editWindow, image=button_image_3, borderwidth=0, highlightthickness=0, command=modifierEtudiant, relief="flat")
    submitBtn.place(x=28.0, y=402.0, width=96.0, height=25.0)
    button_image_4 = PhotoImage(master=editWindow, file=editWindowAssets("button_4.png"))
    cancelBtn = Button(master=editWindow, image=button_image_4, borderwidth=0, highlightthickness=0,command=lambda: closeWindow(editWindow),
                       relief="flat")
    cancelBtn.place(x=137.0, y=402.0, width=96.0, height=25.0)
#On empeche le dimensionnement de ce menu de modification par l'utilisateur
    editWindow.resizable(False, False)
    #On lance le menu
    editWindow.mainloop()
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderViewWindow() qui affiche une fenêtre **de visualisation des données d'un étudiant.**

```
#Visualisation des données de l'étudiant séléctionné
def renderViewWindow(selectionDonneeEtudiant):
    #On recupère les données des cellules de la ligne séléctionnée
    matricule=selectionDonneeEtudiant[0]
    nom=selectionDonneeEtudiant[1]
    postnom=selectionDonneeEtudiant[2]
    telephone=selectionDonneeEtudiant[3]
    email=selectionDonneeEtudiant[4]
    addresse=selectionDonneeEtudiant[5]

    conn = connection()
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM etudiants WHERE matricule='{matricule}' ")
    resultat = cursor.fetchone()
    conn.commit()
    conn.close()
#On crée une fenêtre pour afficher les données d'une ligne séléctionnée d'un étudiant
#N.B: le matriculeInput, nomInput, etc. c'est pour afficher ces données sur des zones des textes bien que l'on va pas les modifier
    print('Rendu de la fenêtre de visualisation des données')
    viewWindow = Tk()
    viewWindow.title('Fenêtre de visualisation Etudiant- Système de gestion des profiles des étudiants')
    viewWindow.geometry("720x480")
    viewWindow.configure(bg = "#FFFFFF")
    viewCanvas = Canvas(viewWindow,bg = "#FFFFFF",height = 480,width = 720,bd = 0,highlightthickness = 0,relief = "ridge")
    viewCanvas.place(x = 0, y = 0)
    image_image_1 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_1.png"))
    viewCanvas.create_image(360.0, 264.0, image=image_image_1)
    image_image_2 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_2.png"))
    viewCanvas.create_image(360.0, 24.0, image=image_image_2)
    viewCanvas.create_text(49.0, 10.0, anchor="nw", text="Visualiser l'Etudiant", fill="#FFFFFF", font=("Inter SemiBold", 24 * -1))
    image_image_3 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_3.png"))
    viewCanvas.create_image(28.0, 24.0, image=image_image_3)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderViewWindow() qui affiche une fenêtre **de visualisation des données d'un étudiant. (Cont.)**

```
#On déclare les input zones qui vont afficher les données des cellules de la ligne séléctionnée
    #input matricule
    image_image_5 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_5.png"))
    viewCanvas.create_image(456.0, 104.0, image=image_image_5)
    entry_image_2 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_2.png"))
    viewCanvas.create_image(455.0, 109.5, image=entry_image_2)
    matriculeInput = Entry(master=viewWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    matriculeInput.place(x=325.0, y=98.0, width=260.0, height=21.0)
    #matriculeInput.bind("<Key>", lambda e: "break") lie un événement à l'élément graphique nomInput, généralement un champ de saisie (Entry),
    #de sorte que lorsque n'importe quelle touche est pressée ("<Key>"),
    #l'événement est intercepté par la fonction lambda qui renvoie "break".
    #En tkinter, "break" est utilisé pour arrêter la propagation de l'événement.
    #Dans notre contexte, cela signifie que lorsque l'utilisateur essaie de saisir quelque chose dans le champ matriculeInput,
    #l'événement de touche est capturé mais ne produit aucun effet visible, car la saisie est immédiatement interrompue par le "break",
    #empêchant ainsi la modification du contenu du champ de saisie.
    #CCela peut être utilisé pour rendre un champ de saisie en lecture seule, par exemple.
    matriculeInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(325.0, 85.0, anchor="nw", text="Matricule", fill="#000000", font=("Inter", 11 * -1))
    matriculeInput.insert(0,matricule)
    #input nom
    image_image_4 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_4.png"))
    viewCanvas.create_image(457.0, 166.0, image=image_image_4)
    entry_image_1 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_1.png"))
    viewCanvas.create_image(456.0, 171.5, image=entry_image_1)
    nomInput = Entry(master=viewWindow,bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    nomInput.place(x=326.0, y=160.0, width=260.0, height=21.0)
    nomInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(326.0, 147.0, anchor="nw", text="Nom", fill="#000000", font=("Inter", 11 * -1))
    nomInput.insert(0,nom)
    #input postnom
    image_image_6 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_6.png"))
    viewCanvas.create_image(457.0,230.0,image=image_image_6)
    entry_image_3 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_3.png"))
    viewCanvas.create_image(456.0,235.5,image=entry_image_3)
    postnomInput = Entry(master=viewWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    postnomInput.place(x=326.0,y=224.0,width=260.0,height=21.0)
    postnomInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(326.0,211.0,anchor="nw",text="Post Nom",fill="#000000",font=("Inter", 11 * -1))
    postnomInput.insert(0,postnom)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit une fonction renderViewWindow() qui affiche une fenêtre **de visualisation des données d'un étudiant. (Cont.)**

```
    #input telephone
    image_image_7 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_7.png"))
    viewCanvas.create_image(166.0,294.0,image=image_image_7)
    entry_image_4 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_4.png"))
    viewCanvas.create_image(165.0,299.5,image=entry_image_4)
    telephoneInput = Entry(master=viewWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    telephoneInput.place(x=35.0,y=288.0,width=260.0,height=21.0)
    telephoneInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(35.0,275.0,anchor="nw",text="Téléphone",fill="#000000",font=("Inter", 11 * -1))
    telephoneInput.insert(0,telephone)
    #input email
    image_image_8 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_8.png"))
    viewCanvas.create_image(457.0,294.0,image=image_image_8)
    entry_image_5 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_5.png"))
    viewCanvas.create_image(456.0,299.5,image=entry_image_5)
    emailInput = Entry(master=viewWindow, bd=0,bg="#FFFFFF",fg="#000716",highlightthickness=0)
    emailInput.place(x=326.0,y=288.0,width=260.0,height=21.0)
    emailInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(326.0,275.0,anchor="nw",text="Email",fill="#000000",font=("Inter", 11 * -1))
    emailInput.insert(0,email)
    #input addresse
    image_image_9 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_9.png"))
    viewCanvas.create_image(311.0, 358.0, image=image_image_9)
    entry_image_6 = PhotoImage(master=viewWindow, file=viewWindowAssets("entry_6.png"))
    viewCanvas.create_image(310.5, 363.5, image=entry_image_6)
    addresseInput = Entry(master=viewWindow, bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
    addresseInput.place(x=35.0, y=352.0, width=551.0, height=21.0)
    addresseInput.bind("<Key>", lambda e: "break")
    viewCanvas.create_text(35.0,339.0,anchor="nw",text="Addresse",fill="#000000",font=("Inter", 11 * -1))
    addresseInput.insert(0,addresse)

    image_image_10 = PhotoImage(master=viewWindow, file=viewWindowAssets("image_10.png"))
    viewCanvas.create_image(166.0, 167.0, image=image_image_10)
    image_image_11 = PhotoImage(master=viewWindow, file=f"./assets/uploaded/{resultat[1]}")
    viewCanvas.create_image(168.0, 168.0, image=image_image_11)
#On bloque le redimensionnement de la fenêtre
    viewWindow.resizable(False, False)
    #On lance la fenêtre
    viewWindow.mainloop()
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit la fenêtre principale de notre application

```
#On crée le menu principal de l'application
mainWindow = Tk()
mainWindow.title('Accueil - Système de gestion des profiles des Etudiants')
mainWindow.geometry("1080x720")
mainWindow.configure(bg = "#FFFFFF")
mainCanvas = Canvas(mainWindow,bg = "#FFFFFF",height = 720,width = 1080,bd = 0,highlightthickness = 0,relief = "ridge")
mainCanvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(file=mainWindowAssets("image_1.png"))
image_1 = mainCanvas.create_image(645.0,397.0,image=image_image_1)
image_image_2 = PhotoImage(file=mainWindowAssets("image_2.png"))
image_2 = mainCanvas.create_image(648.0,398.0,image=image_image_2)
image_image_3 = PhotoImage(file=mainWindowAssets("image_3.png"))
image_3 = mainCanvas.create_image(540.0,37.0,image=image_image_3)
mainCanvas.create_text(73.0,15.0,anchor="nw",text="Système de Gestion des Profiles des Etudiants",fill="#FFFFFF",font=("Inter SemiBold", 36 * -1))
image_image_4 = PhotoImage(file=mainWindowAssets("image_4.png"))
image_4 = mainCanvas.create_image(38.0,36.0,image=image_image_4)
image_image_5 = PhotoImage(file=mainWindowAssets("image_5.png"))
image_5 = mainCanvas.create_image(105.0,397.0,image=image_image_5)
button_image_1 = PhotoImage(file=mainWindowAssets("button_1.png"))
#On charge les images des boutons
button_1 = Button(image=button_image_1,borderwidth=0,highlightthickness=0,command=renderAddWindow,relief="flat")
button_1.place(x=28.0,y=105.0,width=148.0,height=57.0)
button_image_2 = PhotoImage(file=mainWindowAssets("button_2.png"))
button_2 = Button(image=button_image_2,borderwidth=0,highlightthickness=0,command=modifierEtudiant,relief="flat")
button_2.place(x=28.0,y=187.0,width=148.0,height=57.0)
button_image_3 = PhotoImage(file=mainWindowAssets("button_3.png"))
button_3 = Button(image=button_image_3,borderwidth=0,highlightthickness=0,command=supprimerEtudiant,relief="flat")
button_3.place(x=28.0,y=269.0,width=148.0,height=57.0)
button_image_4 = PhotoImage(file=mainWindowAssets("button_4.png"))
button_4 = Button(image=button_image_4,borderwidth=0,highlightthickness=0,command=voirEtudiant,relief="flat")
button_4.place(x=28.0,y=351.0,width=148.0,height=57.0)
button_image_5 = PhotoImage(file=mainWindowAssets("button_5.png"))
button_5 = Button(image=button_image_5,borderwidth=0,highlightthickness=0,command=exporterExcel,relief="flat")
button_5.place(x=28.0,y=433.0,width=148.0,height=57.0)
```

Gestion des profiles des étudiants (Cont.)

- Ce code définit la fenêtre principale de notre application (Cont.)

```
#On crée une nouvelle instance de la classe Style de ttk pour gérer les styles des widgets Treeview.
style = ttk.Style()
#On configure un style appelé "mystyle.Treeview" appelé en haut pour le widget Treeview.
#Les options configurées sont :
    ##highlightthickness=0: On supprime la mise en évidence autour du widget lorsqu'il a le focus.
    ##bd=0: On supprime la bordure du widget.
    ##font=('Inter SemiBold', 12): On définit la police du texte du widget.
    ##rowheight=30: On définit la hauteur des lignes du widget.
style.configure("mystyle.Treeview", highlightthickness=0, bd=0, font=('Inter SemiBold', 12),rowheight=30)
#On configure un style appelé "mystyle.Treeview.Heading" pour les en-têtes de colonnes du Treeview.
#Les options configurées sont :
    ##font=('Inter SemiBold', 12,'bold'): Définit la police en gras pour les en-têtes.
    ##background="black": Définit la couleur de fond des en-têtes à noir.
    ##foreground='black': Définit la couleur du texte des en-têtes à noir.
style.configure("mystyle.Treeview.Heading", font=('Inter SemiBold', 12,'bold'),background="black",foreground='black')
#On configure le layout du Treeview pour supprimer les bordures.
#Cela signifie que seul la partie de l'arbre du Treeview sera affichée sans bordures.
#'mystyle.Treeview.treearea': Cela fait référence à la partie de l'arbre du Treeview
#qui contient les éléments hiérarchiques (les lignes du Treeview). C'est la zone principale du Treeview où les données sont affichées.

#{'sticky': 'nswe'}: C'est une configuration supplémentaire pour le layout.
#Dans ce cas, 'sticky': 'nswe' signifie que la zone de l'arbre ('mystyle.Treeview.treearea') doit s'étirer dans toutes
#les directions (nord, sud, ouest, est), remplissant ainsi tout l'espace disponible dans le Treeview

style.layout("mystyle.Treeview", [('mystyle.Treeview.treearea', {'sticky': 'nswe'})])
#On raffraichit les données
renderTreeVIew(LireLesDonneesDeLaBDD())
#On empêche le redimensionnement du menu principal
mainWindow.resizable(False, False)
#On lance le menu principal
mainWindow.mainloop()
```