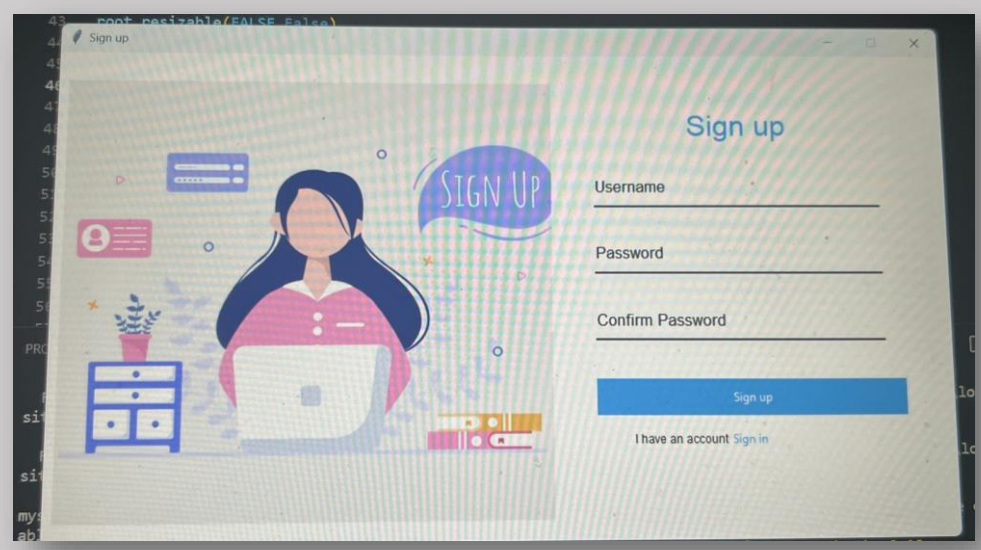
Rapport projet :

L'interface d’inscription




```
def Inscription():
    user=txtuser.get()
    password=txtmot.get()
    password_conf=txtmotcon.get()

    if(user==""or password=="" or password_conf==""):
        messagebox.showerror("","il faut saisir tous les informations")
    elif (password!=password_conf):
        messagebox.showerror("","mot de passe incorrect")
    else:
        try:
            maBase=mysql.connector.connect(host="localhost",user="root",password="",database="bibliothèque")
            meConnect=maBase.cursor()
            sql="select *from utilisateur where username='{}'".format(txtuser.get())
            meConnect.execute(sql)
            rows=meConnect.fetchone()
            if rows!=None:
                messagebox.showerror("Erreur","Ce user existe déja")
            else:
                sql="INSERT INTO utilisateur(username,password,password_confirma) VALUES (%s,%s,%s)"
                val=(user,password,password_conf)
                meConnect.execute(sql,val)
                maBase.commit()
                call(["python","login.py"])


        except Exception as e:
            print(e)
            maBase.rollback()
            maBase.close()
```

La fonction **Inscription()** est appelée lorsqu'on clique sur le bouton "Valider" . Elle récupère les données saisies par l'utilisateur (nom d'utilisateur, mot de passe, confirmation du mot

de passe) à l'aide des champs de texte **txtuser**, **txtmot** et **txtmotcon**.

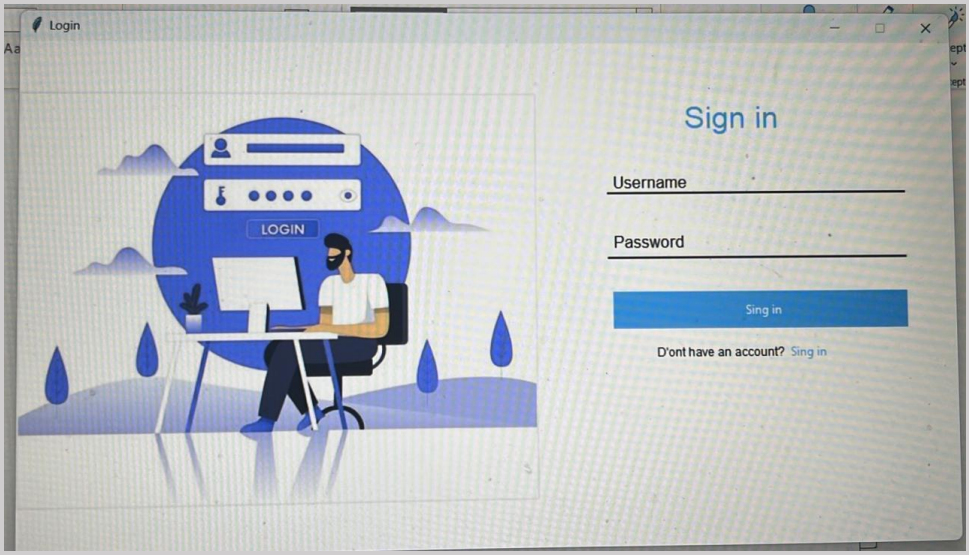
Ensuite, la fonction vérifie si tous les champs sont remplis et si le mot de passe est identique dans les deux champs de mot de passe. Si un des deux cas n'est pas vérifié, une boîte de dialogue affiche un message d'erreur correspondant.

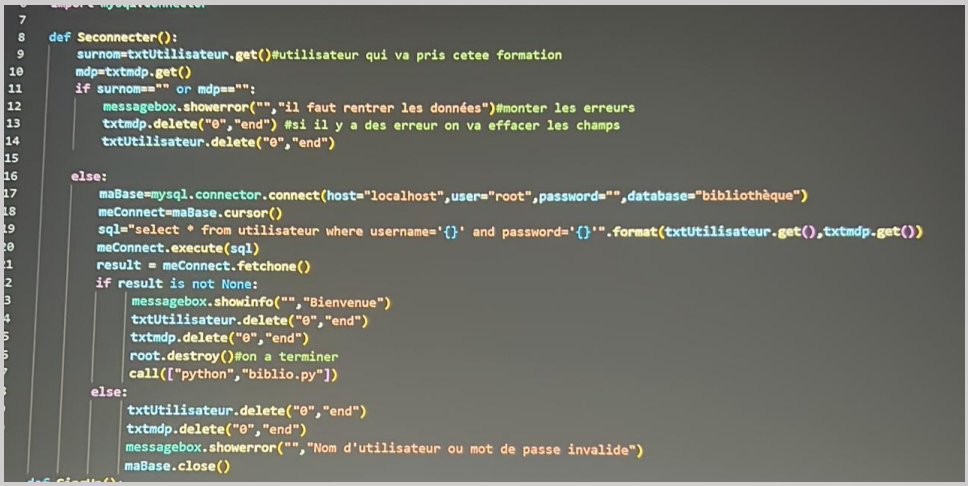
Si les champs sont correctement remplis, la fonction se connecte à la base de données MySQL en utilisant les informations d'identification de l'utilisateur (**localhost**, **root**, ``, **bibliothèque**) et insère les données de l'utilisateur dans la table **utilisateur** en utilisant une requête SQL **INSERT**. En cas de succès, la fonction ferme la fenêtre d'inscription et ouvre une nouvelle fenêtre pour permettre à l'utilisateur de se connecter. En cas d'erreur, elle affiche un message d'erreur et annule la transaction

```
7   def connexion():
8       call(["python","signin.py"])
9
```

Il semble que cette fonction **connexion()** soit destinée à ouvrir une fenêtre de connexion. Elle utilise la fonction **call()** du module **subprocess** pour lancer le script **signin.py**. Cela peut être utile si **signin.py** est un script séparé qui contient une interface graphique pour la connexion et que vous souhaitez lancer cette interface à partir de l'interface principale

Interface du Login





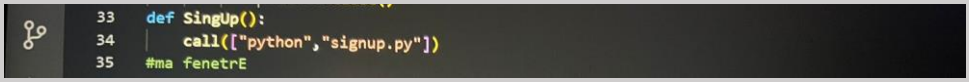
```
def Seconnecter():
    surnom=txtUtilisateur.get()#utilisateur qui va pris cetee formation
    mdp=txtmdp.get()
    if surnom=="" or mdp=="":
        messagebox.showerror("","il faut rentrer les données")#monter les erreurs
        txtmdp.delete("0","end") #si il y a des erreur on va effacer les champs
        txtUtilisateur.delete("0","end")

    else:
        maBase=mysql.connector.connect(host="localhost",user="root",password="",database="bibliothèque")
        meConnect=maBase.cursor()
        sql="select * from utilisateur where username='{}' and password='{}'".format(txtUtilisateur.get(),txtmdp.get())
        meConnect.execute(sql)
        result = meConnect.fetchone()
        if result is not None:
            messagebox.showinfo("","Bienvenue")
            txtUtilisateur.delete("0","end")
            txtmdp.delete("0","end")
            root.destroy()#on a terminer
            call(["python","biblio.py"])
        else:
            txtUtilisateur.delete("0","end")
            txtmdp.delete("0","end")
            messagebox.showerror("","Nom d'utilisateur ou mot de passe invalide")
            maBase.close()
```

Lorsque l'utilisateur appuie sur le bouton "Se connecter" sur l'interface utilisateur, cette fonction est appelée et effectue les étapes suivantes :

1. Elle récupère les informations saisies par l'utilisateur (nom d'utilisateur et mot de passe) à partir de deux champs de texte nommés "txtUtilisateur" et "txtmdp".
2. Elle vérifie que les champs ne sont pas vides. Si l'un ou l'autre est vide, elle affiche une boîte de dialogue d'erreur demandant à l'utilisateur de remplir tous les champs. Elle efface également les champs pour permettre une nouvelle saisie.
3. Si les champs sont remplis, elle se connecte à une base de données de bibliothèque en utilisant les informations de connexion suivantes : nom d'utilisateur = "root", mot de passe vide, nom de la base de données = "bibliothèque", et adresse de la base de données = "localhost".
4. Elle exécute une requête SQL qui sélectionne toutes les entrées dans la table "utilisateur" où le nom d'utilisateur et le mot de passe correspondent aux valeurs saisies par l'utilisateur.
5. Elle récupère la première entrée (ou aucune) de la réponse de la base de données en utilisant la méthode "fetchone()" de l'objet "meConnect".
6. Si une entrée est trouvée, cela signifie que les informations de connexion sont correctes, elle affiche une boîte de dialogue de bienvenue et efface les champs pour permettre une nouvelle saisie. Elle ferme ensuite la fenêtre de

connexion actuelle et ouvre une nouvelle fenêtre en appelant le script Python " biblio.py”

7. Si aucune entrée n'est trouvée, cela signifie que les informations de connexion sont incorrectes. Elle affiche une boîte de dialogue d'erreur demandant à l'utilisateur de saisir à nouveau le nom d'utilisateur et le mot de passe. Elle efface également les champs pour permettre une nouvelle saisie. Elle ferme également la connexion à la base de données en utilisant la méthode "close()" de l'objet "maBase"

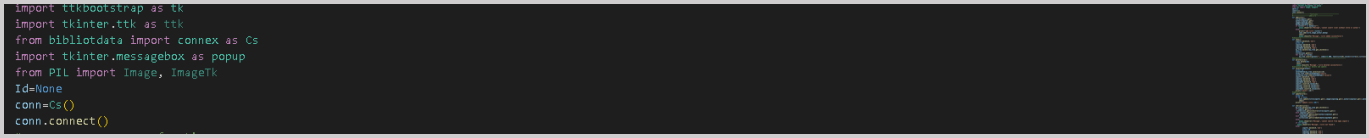


```
33  def SingUp():
34      call(["python","signup.py"])
35  #ma fenetrE
```

La fonction **SingUp()** est appelée lorsqu'on clique sur le bouton "S'inscrire" et elle ouvre une nouvelle fenêtre pour permettre à l'utilisateur de s'inscrire. Elle utilise la fonction **call()** du module **subprocess** pour lancer le script "signup.py" qui contient le code pour gérer l'inscription de l'utilisateur.

Importation des bibliothèques qu’on va travailler avec :

Interface livre

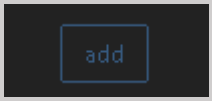


```
import ttkbootstrap as tk
import tkinter.ttk as ttk
from bibliotdata import connex as Cs
import tkinter.messagebox as popup
from PIL import Image, ImageTk
Id=None
conn=Cs()
conn.connect()
```

Pour les fonctions :

```
#------------------functions----------------------
#------------------addlivre-------------------
def addlivre():
    livre=inputti.get()
    auteur=inputaut.get()
    image=inputimg.get()
    datep=inputdate.get()
    if livre=="" or auteur=="":
        popup.showerror('Message','cannot insert liver without title or auteur')
    else:
        print(f"add livre {livre}")
        conn.add(livre,image,auteur,datep)
        show()
        popup.showinfo('Message','livre added successfully')
#showlivre-----------------------------------------
def show():
    inputti.delete(0,'end')
    global ID
    inputaut.delete(0,'end')
    inputimg.delete(0,'end')
    inputdate.delete(0,'end')
    my_tree.delete(*my_tree.get_children())
    ID=None
    livres=conn.getAll()
    for livre in livres:
        my_tree.insert(parent="", index=tk.END, text=livre[0],values=(livre[1],livre[2],livre[3],livre[4]))
#deletelivre-----------------------------------------------------
def deletelivre():
    conn.delete(ID)
    show()
    popup.showinfo('Message','livre deleted successfully')
#selection of item for delete and update:
def selectItem(event):
    global ID
    seletedItem=my_tree.selection()[0]
    ID=my_tree.item(seletedItem)["text"]
    values=my_tree.item(seletedItem)["values"]
    inputti.delete(0,'end')
    inputaut.delete(0,'end')
    inputimg.delete(0,'end')
    inputdate.delete(0,'end')
    inputti.insert(0,values[0])
    inputaut.insert(0,values[2])
    inputimg.insert(0,values[1])
    inputdate.insert(0,values[3])
    print(f"select {id}")
```

- **Addlivre :**
  Premièrement je récupère les données dans les inputs avec un test si les inputs du titre et auteur sont vide si oui en affiche un message d'erreur parce que on peut pas ajouter un livre sans titre ou auteur. Cette méthode est utilisée par la Button add.



Cette fonction utilise une autre méthode add qui injecter une requête MySQL avec les valeurs dans les inputs ADD :

```
    def add(self,titre,image,auteur,annedepub):
        req=f"insert into {self.table}(titre,image,auteur,annedepub)values(%s,%s,%s,%s)"
        values=(titre,image,auteur,annedepub)
        self.Cursor.execute(req,values)
        self.conn.commit()
```

- **Show() :**
  Premièrement elle vide tous les inputs puis on fait appel

```
my_tree.delete(*my_tree.get_children())
```

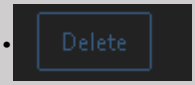
cela permet de vider les tables dans l'affichage si il 'Ya quelque chose afficher déjà puis je stock dans une variables « livres » le return du fonction :

```
conn.getAll()
   def getAll(self):
        req=f"select * from {self.table}"
        self.Cursor.execute(req)
        livres=self.Cursor.fetchall()
        return livres
```

une méthode qui fait injecter avec MySQL une requête « select* from livres » pour récupère tous les livres dans la base de données puis je boucle pour chaque livre et j'insert leur information avec insert().

- **Deletelivre() :**
  Il fait appel a :

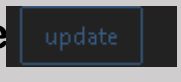
```
conn.delete(ID)
 def delete(self,id):
        req=f"delete from {self.table} where id=%s"
        values=(id,)
        self.Cursor.execute(req,values)
        self.conn.commit()
        print("deletete succesfully")
```

une méthode qui fait injecter une requête MySQL qui supprime un livre, le recherche est fait par « ID» du livre puis j'appel la fonction show() pour afficher les livres restant dans la table après suppression avec un message pour indiquer que l'opération a passé bien.
Cette fonction est utilise par la Button delete. 

- **SelectItem(event) :**
  Fonction qui permet de récupérer les informations de chaque livre sélectionner dans la table est les stocker dans les inputs convenables.
- **Updatelivre :**
  Fonction pour modifier les informations d'un livre :
  Premièrement on fait un test si déjà ID est sélectionner pour savoir quel livre on veut le modifier puis on appel :

```
conn.update(titre=inputti.get(),image=inputimg.get(),auteur=inputaut.get(),datep=inputda
te.get(),id=ID)
  def update(self,titre,image,auteur,datep,id):
        req=f"update {self.table} set titre=%s,image=%s,auteur=%s,annedepub=%s where
id=%s"
        values=(titre,image,auteur,datep,id)
        self.Cursor.execute(req,values)
        self.conn.commit()
```

qui une méthode qui injecter un requête MySQL, les valeurs injecter sont les valeurs récupérer a partir des inputs ;

cette méthode est utiliser par la Button update 

- **Chercherlivre() :**
  Fonctionne qui permet de rechercher a spécifique livre et l'afficher.
  Le recherche est fait par trois cas avec priorité pour le rechercher par titre :
  Si l'input de titre est vide alors je passe a l'input d'auteur si aussi l'Est vide je passe a la date sinon j'affiche un message d'erreur que le recherche ne peut pas être fait sans l'un des informations précèdent.
  Cette fonction appel ensemble des méthodes selon le type de recherche :
  Pour titre :

```
#get specific livre by titre=======================
    def getlivretitre(self,titre):
        req=f"select * from {self.table} where titre like %s"
        values=(titre+'%',)
        self.Cursor.execute(req,values)
        livres=self.Cursor.fetchall()
        return livres
```

  pour auteur :

```
   def getlivreaut(self,auteur):
        req=f"select * from {self.table} where auteur like %s"
        values=(auteur+'%',)
        self.Cursor.execute(req,values)
        livres=self.Cursor.fetchall()
        return livres
```

  pour date :

```
   def getlivredate(self,date):
        req=f"select * from {self.table} where annedepub like %s"
        values=(date+'%',)
        self.Cursor.execute(req,values)
        livres=self.Cursor.fetchall()
        return livres
```

  ❖ **Pour l'interface :**
  J'ai utiliser tkinter avec ttkboostrap pour la customisation,
  Ttk pour treeview(table) est des simples inputs est labels :

```python
#interface-------------------------------------------------------------------------
window=tk.Window(themename="darkly")
window.geometry("1920x550")
window.title("library")
#image=Image.open("library.png")
#bck_end=ImageTk.PhotoImage(image)
#lbl=tk.Label(window,image=bck_end)
#lbl.place(x=0,y=0)
#=========================================================================#
s=tk.Style()
#----------------------table----------------------------
s.configure('Treeview', rowheight=50)
my_tree=tk.Treeview(window,columns=("titre","image","auteur","datedepub"),height=50)
my_tree.heading("#0",text="ID")
my_tree.heading("titre",text="Titre")
my_tree.heading("image",text="Image")
my_tree.heading("auteur",text="Auteur")
my_tree.heading("datedepub",text="Date de pub")
my_tree.column("#0",width=50)
my_tree.column("titre",width=200)
my_tree.column("image",width=100)
my_tree.column("auteur",width=200)
my_tree.column("datedepub",width=100)
my_tree.pack(side=tk.LEFT,fill=tk.BOTH,expand=True)
#---------------greeting----------------------
greeting = tk.Label(text="welcome to our library",bootstyle="primary")
greeting.pack(pady=20)
#---------------chercher----------------------------
#--------------------titre-------------------------------
titelab=tk.Label(text="titre")
titelab.pack(padx=100,pady=10)
inputti=tk.Entry(window)
inputti.pack(side=tk.TOP)
#----------------------auteur--------------------
auteurlab=tk.Label(text="Auteur")
```