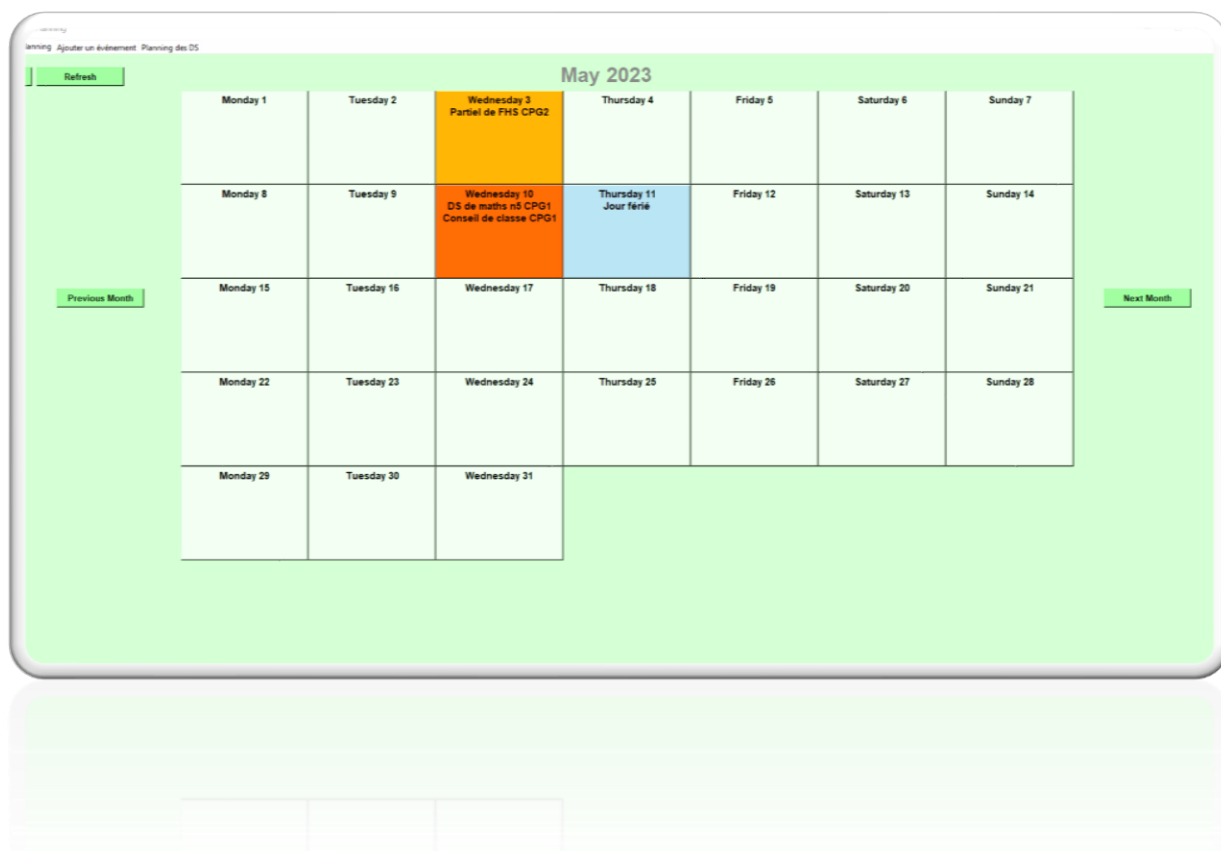


APPLICATION DE PLANNING



SOMMAIRE

1) Présentation de l'application (p2)

2) Cahier des charges (p6)

3) Moyens utilisés (p7)

4) Création de l'application

- a. Création de la base de données (p8)**
- b. Partie technique, code python (p10)**
 - i. Création de la fenêtre**
 - ii. Récupération des informations**
 - iii. Ajouter un événement**
 - iv. Planning et mise en forme**
- c. Conversion en exécutable (p19)**

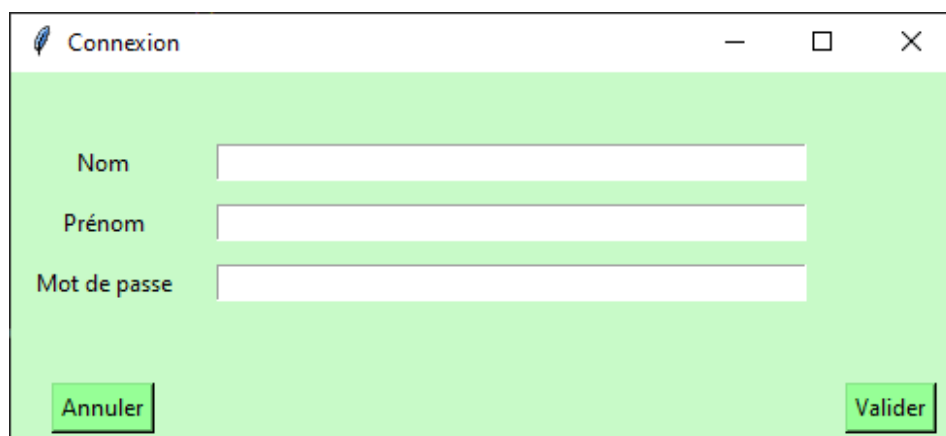
5) Déroulement du projet (p20)

6) Conclusion (p21)

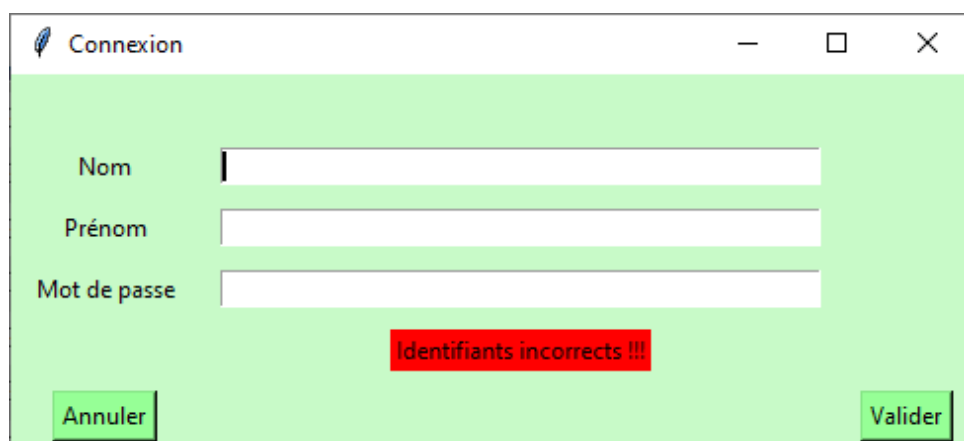
1) PRESENTATION DU PROJET

Nous avons réalisé un planning qui va montrer à l'utilisateur tous les événements du mois qui le concerne.

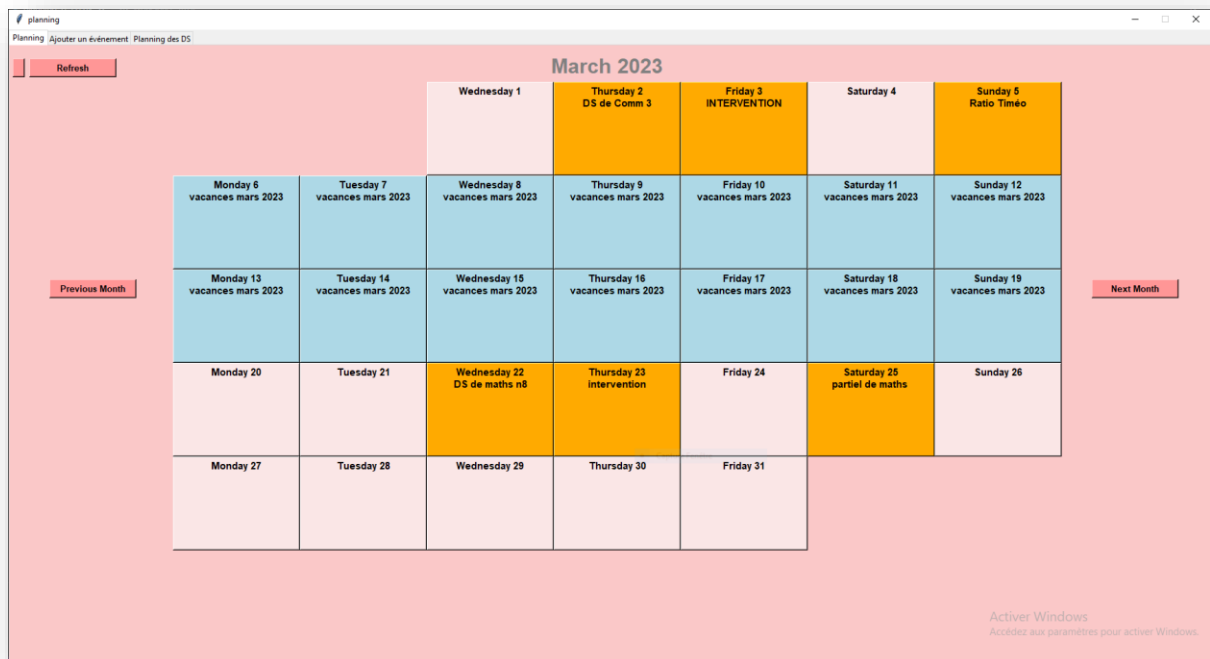
Quand on lance l'application **une fenêtre s'ouvre** :



Ici, on peut rentrer notre **nom, prénom et mot de passe**. Si les 3 conditions ne concordent pas avec les données stockées dans la base de données, **l'application ne se lance pas**, on ne peut pas y accéder :



On accède ensuite à l'**application** en elle-même qui donne ceci :



Ce **1^{er} onglet** montre à l'utilisateur **les événements qui le concernent**. Si l'utilisateur est **un élève** ou **un administrateur**, il ne verra pas la même chose. L'administrateur peut voir l'ensemble des événements.

Si l'utilisateur est un administrateur il aura la possibilité **d'ajouter un événement** qui se trouve au niveau **du 2^{ème} onglet** de la fenêtre. On arrive sur cette page :

The screenshot shows the 'Ajouter un événement' (Add Event) form. The form has a light blue background and contains the following fields and controls:

- Titre**: Text input field.
- Debut**: Text input field.
- Fin**: Text input field.
- Catégorie**: Dropdown menu.
- Groupe**: Dropdown menu.
- Salle**: Text input field.
- Description**: Text input field.
- Ajouter une personne:** Text input field with an **Ajouter la personne** button next to it.
- ☐ **salle réservée**: Checkbox.
- Ajouter l'événement**: Button.

Ici, on peut **ajouter un événement** qui sera ensuite **affiché sur le planning** en rentrant les informations correspondantes sur les lignes.

Il faut noter que le fait d'ajouter **des événements** n'est **disponible que pour les administrateurs**.

Categorie	<input type="text"/>
Groupe	<input type="text"/>
Salle	<input type="text"/>

Groupe	<input type="text"/>
Salle	<input type="text"/>
Description	<input type="text"/>
er une personne:	<input type="text"/>

« **combobox** » permettant de choisir un choix parmi ceux proposés ici pour la **catégorie** et la **classe(groupe)**.

Sur le **3^{ème} onglet** maintenant, on a un planning des DS pour les **CPG1** et les **CPG2** qu'on peut cliquer pour avoir leur **planning des DS**.

Planning

Ajouter un événement

Planning des DS

CPG1

CPG2

Date

23/03/2023

02/04/2023

03/04/2023

03/05/2023

10/05/2023

10/05/2023

Horaire

10:00

10:00

10:00

20:00

00:00

00:00

Titre

intervention

à tester

tire

svdsfv

zraqzsh

bbsdbfb

Salle

Plusieurs salles

Plusieurs salles

Plusieurs salles

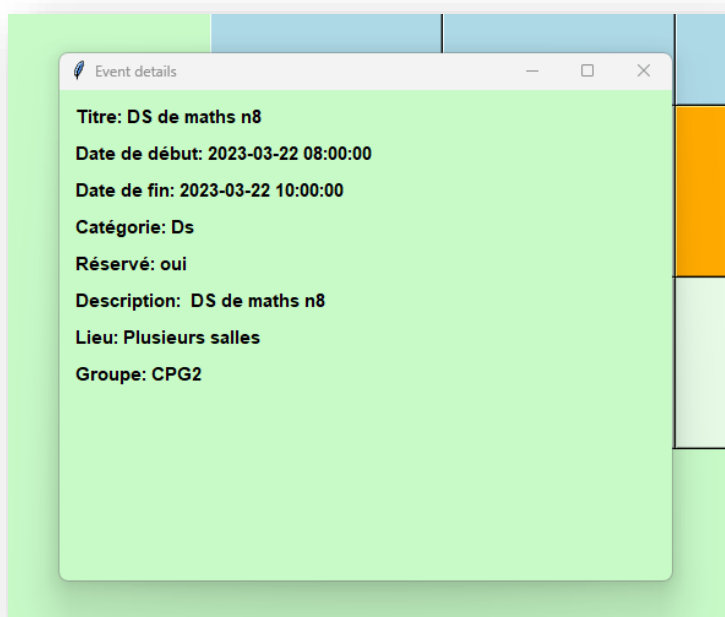
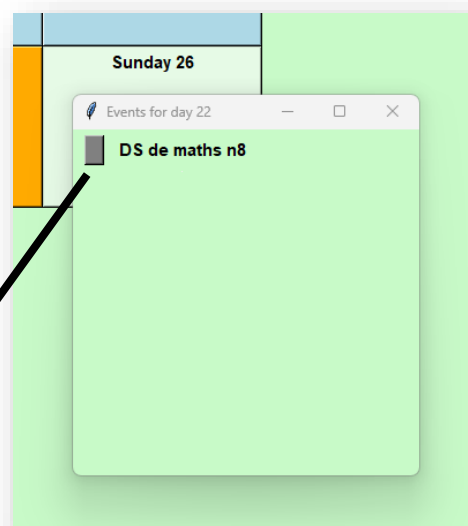
Plusieurs salles

C406

C406

Dans le **1^{er} onglet** on peut trouver les **détails** concernant un événement en cliquant sur le jour puis l'événement souhaité :

March 2023						
		Wednesday 1	Thursday 2 DS de Conna 3	Friday 3	Saturday 4	Sunday 5
Monday 6 vacances mars 2023	Tuesday 7 vacances mars 2023	Wednesday 8 vacances mars 2023	Thursday 9 vacances mars 2023	Friday 10 vacances mars 2023	Saturday 11 vacances mars 2023	Sunday 12 vacances mars 2023
Monday 13 vacances mars 2023	Tuesday 14 vacances mars 2023	Wednesday 15 vacances mars 2023	Thursday 16 vacances mars 2023	Friday 17 vacances mars 2023	Saturday 18 vacances mars 2023	Sunday 19 vacances mars 2023
Monday 20	Tuesday 21	Wednesday 22 DS de maths n8	Thursday 23	Friday 24	Saturday 25 partiel de maths	Sunday 26
Monday 27	Tuesday 28	Wednesday 29	Thursday 30	Friday 31		



2) CAHIER DES CHARGES DU PROJET

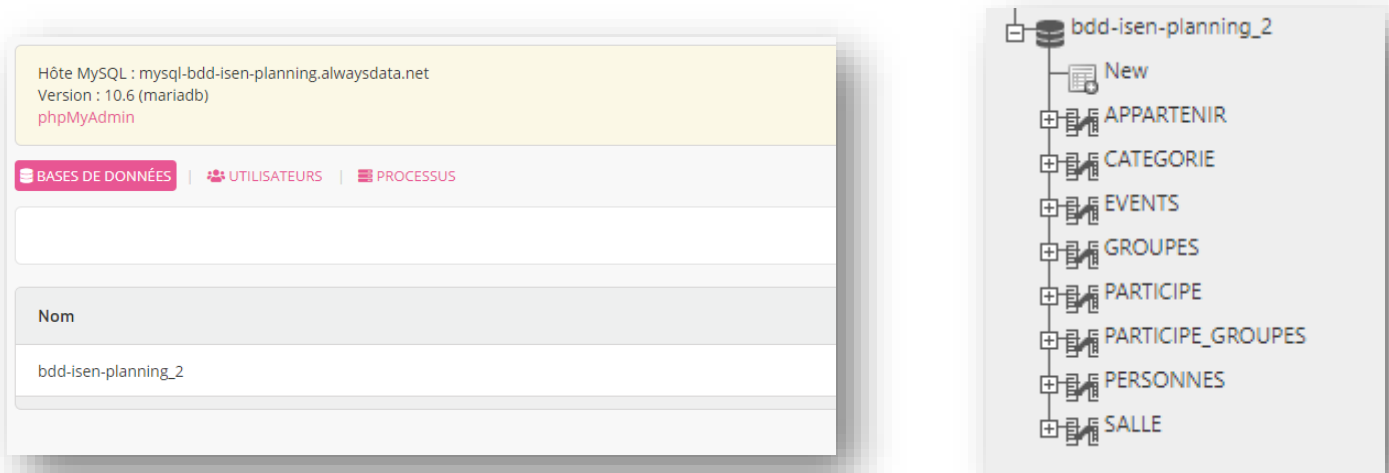
Sujet : Création d'une application de planning pour les classes de CPG ISEN Lille. Cette application vise une utilisation de la part des élèves et des enseignants, elle devra récapituler les événements importants tels que les DS, partiels, conseils de classe, vacances, présentations, etc. Chaque personne devra pouvoir se connecter à l'application et voir les événements qui la concerne.

Critères :

- Utilisation d'une base de données (BDD) en ligne afin de centraliser les informations des événements et des personnes
- Mise en place d'une application utilisable sur tout ordinateur par les membres de l'ISEN grâce à leurs identifiants
- Traitement par l'application des informations de la base de données afin d'afficher les informations utiles à l'utilisateur
- Affichage d'un récapitulatif des DS de l'année par promo
- Gestion des permissions en fonction du statut élève/professeur qui permettra de rajouter, modifier ou supprimer des événements

3) MOYENS UTILISES

Premièrement afin de **stocker les données** nous avons créé une base de données en ligne sur un **site d'hébergement**. Nous avons choisi le site **alwaysdata**. Voici ce à quoi ressemble notre base de données (BDD) sur le site :

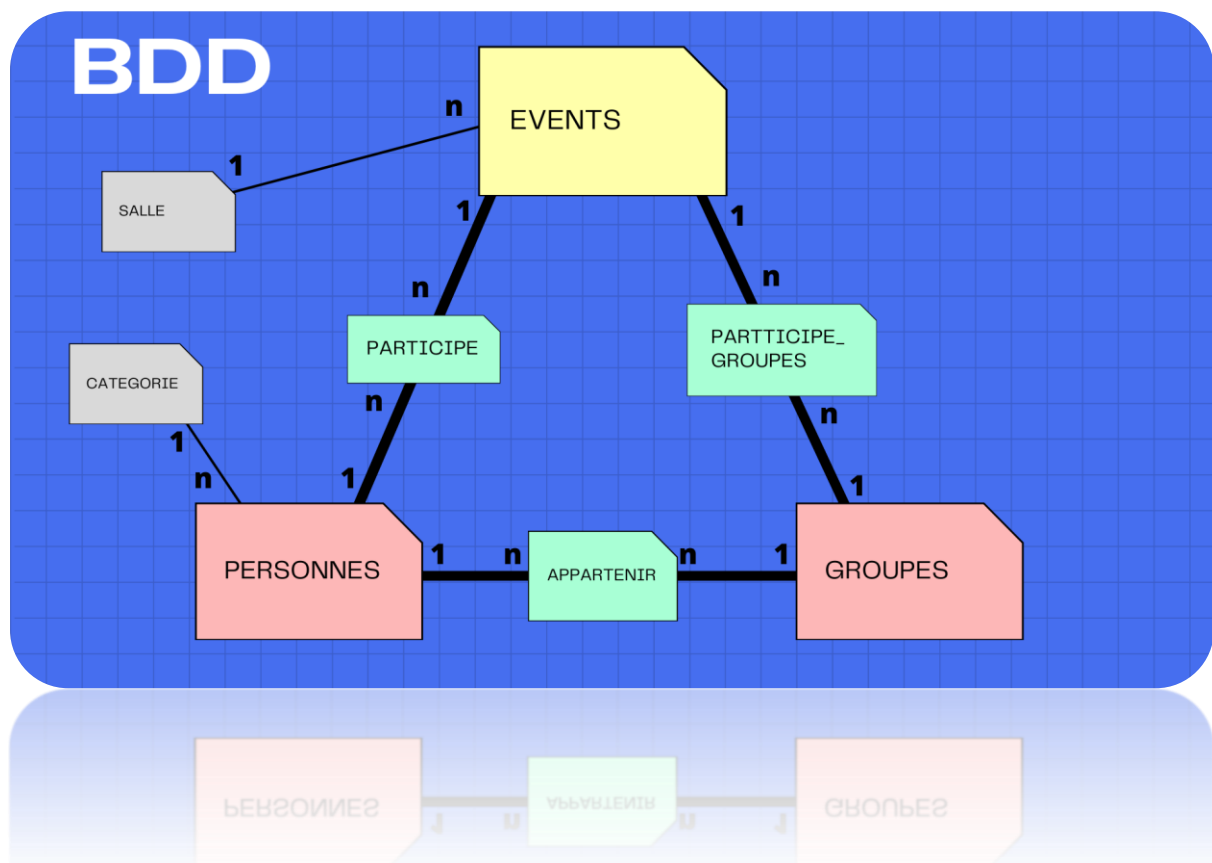


Ensuite pour créer l'application on va d'abord la **coder sous python** pour ensuite la **convertir en exécutable** afin que n'importe qui puisse l'utiliser sur son ordinateur **sans prérequis**. Sous python nous utiliserons le module **tkinter** afin de créer l'affichage, les boutons, les textes, etc. Et le module **mysql** connectera le code avec la base de données. Les informations récupérées grâce à cette connexion seront ensuite utilisées dans l'affichage. Toutes les **fonctionnalités** seront ensuite créées dans le code directement, **la base de données ne sert qu'à stocker des informations** qui sont ensuite **récupérées par le code** lors de son exécution.

4) CREATION DE L'APPLICATION

A. CREATION DE LA BDD

Le MCD (Modèle conceptuel de données) de notre base de données ressemble à ceci :



Les élèves comme les professeurs sont considérés comme des **personnes** et les **permissions** seront gérées directement dans le code de l'application. Les **liens de participation** qui relient une personne ou un groupe de personnes à un événement sont **gérés dans les tables « PARTICIPE » et « PARTICIPE_GROUPS »**

Les liaisons EVENTS-SALLE et CATEGORIE-PERSONNES n'ont pas de tables intermédiaires car on ne peut lier qu'une catégorie une personne, alors que pour la liaison EVENTS-PERSONNES il est nécessaire de rajouter une table car un événement peut faire participer plusieurs personnes qui elles-mêmes ont plusieurs événements dans leur calendrier. **Ces liaisons représentent la « participation »** d'une personne à un événement, il nous faudra donc les **créer** grâce à notre code lorsque l'on voudra **faire participer une personne** ou un groupe **à un événement**.

Chacune de ces tables comportent des **colonnes** qui définissent les **caractéristiques** de **chaque élément** de la table. Voici un exemple pour la table **EVENTS** :

EventID	▲	1	Titre	Date_debut	Date_fin	Categorie	Reserve	Description	Salle_ID_ext
1			DS 5 de Maths CPG2	2023-02-04 13:46:37	2023-02-04 15:46:37	DS	0	bla bla bla	1

Toutes ces **caractéristiques** forment les **informations liées à cet événement** et nous pourrons les **utiliser** et **analyser** par la suite afin de les afficher correctement lorsque c'est nécessaire **dans l'application**.

PersID	Nom	prenom	email	mot_de_passe	CategorieID_ext
1	locqueneux	t	timeo62000@gmail.com	1	2
2	harlet	victorien	vichar31@gmail.com	1234	2

CategorieID	Nom
1	Professeurs
2	Elèves
3	Admins

Autre exemple avec les **personnes** : on a les **noms**, **prénoms** et **mots de passe** qui serviront lors de la **connexion** sur l'application. Mais aussi la **catégorie**, ici les 2 personnes sont des **élèves**, donc leur **ID de catégorie** est 2 car la liaison des tables **PERSONNES-CATEGORIE** relie l'**ID** n°2 à la **catégorie élève**.

B. PARTIE TECHNIQUE, CODE PYTHON

i. Création de la fenêtre

On utilise le **module** « **tkinter** » pour pouvoir créer un **objet** qu'on appelle « **fenetre** » afin de créer une interface graphique. On importe des **modules** pour pouvoir utiliser des **fonctions supplémentaires**. Notamment on importe tkinter et ttk pour les fenêtres, boutons, cadres de tkinter.

```
from tkinter import *
import tkinter.ttk as ttk
from tkinter.font import *
import mysql.connector as MC
from datetime import datetime
import calendar
import pendulum
import colorsys
```

On a aussi importé la bibliothèque « **mysql.connector** ». C'est la bibliothèque la plus importante puisqu'elle va nous permettre de pouvoir **communiquer avec la base de données depuis le code python**.

On importe aussi d'autres modules qui vont nous permettre par exemple de connaître la date exacte du jour actuel quand on se connecte pour pouvoir afficher le mois qui correspond.

```
fenetre = Tk()
fenetre.title('Planning')
fenetre.config(bg=color_bg)
fenetreHeight = 950
fenetreWidth = 1800
fenetre.geometry("{}x{}+50+50".format(fenetreWidth, fenetreHeight))
fenetre.attributes('-fullscreen', False)
fenetre.resizable(width=False, height=False)
```

On crée ensuite l'objet « **fenetre** » qui va nous servir à interagir avec l'application. On choisit ses caractéristiques comme sa dimension et son nom

```
notebook = ttk.Notebook(fenetre)
notebook.grid(row=0,column=0, sticky= 'NW')

# create frames

frame1= Frame(notebook, width=fenetreWidth, height=fenetreHeight, bg=color_bg)
frame2= Frame(notebook, width=fenetreWidth, height=fenetreHeight, bg=color_bg)
frame3= Frame(notebook, width=fenetreWidth, height=fenetreHeight, bg=color_bg)
```

Nous avons également utilisé un objet de tkinter qui s'appelle « **notebook** ». On lui donne la caractéristique de « sticky = 'NW' » pour que les **onglets** soient toujours situés en **haut à gauche de la fenêtre**.

On crée ensuite **3 frames** de notebook qui représentent les **3 onglets** qui nous permettent **d'accéder au planning**, à l'ajout d'un événement et aux **planning récapitulatif de DS**.

ii. Récupération des informations

Pour récupérer les informations qui sont contenus dans la base de données on **connecte le code à la base de données** en utilisant la bibliothèque « mysql.connector ».

```
cnx = MC.connect(user='289285', password='BDDaccess3!')
```

On déclare une variable « cnx » qui est **le lien en ligne avec la base de données**. On utilise un nom d'utilisateur et un mot de passe. Cela nous permettra ensuite de définir les **permissions** des utilisateurs.

```
,host='mysql-bdd-isen-planning.alwaysdata.net',database='bdd-isen-planning_2')
```

On aussi **l'adresse internet** qui nous permet de nous connecter à l'hébergeur du site ainsi que la base de données qui s'appelle « bdd-isen-planning_2 »

Après s'être connectés, on va pouvoir déterminer notre

Requête à la BDD en langage SQL. Une fois que

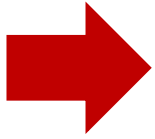
notre requête est enregistrée en tant que variable, on

```
cursor = cnx.cursor()
req = 'SELECT * FROM PERSONNES'
cursor.execute(req)
L = cursor.fetchall()
```

exécute la requête et la méthode « fetchall » nous permet de **récupérer ce que renvoie la BDD** après la requête SQL. Cette requête par exemple, nous permet de voir les personnes qu'il y a dans la table PERSONNES de la BDD et on en récupère une liste. Si on demande à la console python d'afficher la première ligne de cette ligne on obtient :

```
(2, 'harlet', 'victorien', 'vichar31@gmail.com', '1234', 2),
```

Ce qui correspond bien aux lignes qu'il y a dans la BDD qu'on peut voir ci-dessous :



PersID	Nom	prenom	email	mot_de_passe	CategorieID_ext
1	locqueneux	t	timeo62000@gmail.com	1	2
2	harlet	victorien	vichar31@maill.com	1234	2

C'est ainsi qu'on va pouvoir **définir les permissions** des personnes qui se connectent sur l'application. En effet quand elles se connectent, **grâce à la table catégorie**, on va savoir si cette personne est **un élève, un professeur ou un administrateur**.

```
def defPerms ():
    global persCat
    global perms
    if persCat == 1: # prof
        perms = 1
    elif persCat == 2: # élève
        perms = 0
    elif persCat == 3: # admin
        perms = 2
```

Être un administrateur **permet de rajouter des événements** dans la base de données. Donc on affiche l'onglet 2 **uniquement** si l'utilisateur est un **administrateur**.

```
notebook.add(frame1, text='Planning')
if perms == 2:
    notebook.add(frame2, text='Ajouter un événement')
notebook.add(frame3, text='Planning des DS')
```

iii. Ajouter un événement

```
def insert_event_in_bdd():
```

On va maintenant s'intéresser à la fonction « `insert_event_in_bdd()` ».

Celle-ci va nous permettre à partir de l'application d'insérer un événement dans la BDD.

```
def insert_event_in_bdd():
    global count1
    global countPG
    cnx = MC.connect(user='289285', password='BDDaccess3!', host='mysql-bdd-isen-planning.alwaysdata.net', database='bdd-isen-planning_2')
    cursor = cnx.cursor()
    count1+=1
    countPG+=1
    req = "INSERT INTO EVENTS (EventID, Titre, Date_debut, Date_fin, Categorie, Reserve, Description, Salle_ID_ext) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
    infos = (count1, Titre2.get(), Debut2.get(), Fin2.get(), Categorie2.get(), salle_check.get(), Description2.get(), salleget(2))
    cursor.execute(req, infos)
    cnx.commit()
    ajouter_personne2()
    req = 'SELECT Nom FROM GROUPEs'
    cursor.execute(req)
    L = cursor.fetchall()
    for k,j in enumerate(L):
        if Group2.get()==j[0]:
            M=k+1

    if Group2.get()!='JUNIA':
        req = "INSERT INTO PARTICIPE GROUPEs (ParticipeID_g, EventID_ext, ClasseID_ext) VALUES (%s, %s, %s)"
        infos = (countPG, count1, M)
        cursor.execute(req, infos)
        cnx.commit()
        countPG+=1
    else:
        req = "INSERT INTO PARTICIPE GROUPEs (ParticipeID_g, EventID_ext, ClasseID_ext) VALUES (%s, %s, %s)"
        infos = (countPG, count1, 100)
        cursor.execute(req, infos)
        cnx.commit()
        countPG+=1
```

On va dans un premier temps se connecter à la base de données et insérer les informations qui sont rentrées dans les cases par les utilisateurs. Ces cases sont **les variables** avec un 2 à la fin (Titre2, Fin2). On va récupérer les données grâce à **la méthode get()** qui nous donne ce qu'il y a dans les **cases**.

Après avoir effectué notre requête, on utilise la méthode `commit()` qui elle permet de **changer directement** dans la base de données ce qu'on a **modifié**.

```
cursor.execute(req,infos)
cnx.commit()
```

Après avoir mis les changements dans la BDD, on effectue la fonction **ajouter_personne2()** qui va **ajouter les personnes** qu'on a rajoutées à la main dans la BDD aussi.

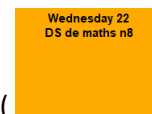
```
ajouter_personne2()
```


Le reste du code de la fonction consiste à rajouter toutes les données dans **PARTICIPE_GROUPES** pour que la BDD fonctionne correctement et qu'on ait **les liens adéquats**.

iv. Planning et mise en forme

Dans le **premier onglet** on a le **planning principal** qui affiche les **événements** concernant l'utilisateur pour le **mois sélectionné** :

March 2023						
		Wednesday 1	Thursday 2 DS de Comm 3	Friday 3	Saturday 4	Sunday 5
Monday 6 vacances mars 2023	Tuesday 7 vacances mars 2023	Wednesday 8 vacances mars 2023	Thursday 9 vacances mars 2023	Friday 10 vacances mars 2023	Saturday 11 vacances mars 2023	Sunday 12 vacances mars 2023
Monday 13 vacances mars 2023	Tuesday 14 vacances mars 2023	Wednesday 15 vacances mars 2023	Thursday 16 vacances mars 2023	Friday 17 vacances mars 2023	Saturday 18 vacances mars 2023	Sunday 19 vacances mars 2023
Monday 20	Tuesday 21	Wednesday 22 DS de maths n8	Thursday 23	Friday 24	Saturday 25 partiel de maths	Sunday 26
Monday 27	Tuesday 28	Wednesday 29	Thursday 30	Friday 31		



Pour afficher tous les **jours du mois** on va créer un gros **bouton**() pour **chaque jour** du mois. **Son texte et sa couleur dépendent des événements du jour**. Les événements concernant l'utilisateur sont récupérés dans la liste « **persEvents** » grâce à la fonction « **getEvents** » :

```
def getEvents():
    global persEvents
    persEvents = []

    if perms == 2:
        persEvents=DBevents
        return

    req = "SELECT EventID FROM EVENTS INNER JOIN PARTICIPE ON EventID=PARTICIPE.EventID_ext WHERE PersID_ext = %s"
    cursor.execute(req, (str(persID),))
    Y = cursor.fetchall()
    for i in range(len(Y)):
        Y[i]=Y[i][0]

    req = "SELECT EventID FROM EVENTS INNER JOIN PARTICIPE_GROUPEES ON EventID=PARTICIPE_GROUPEES.EventID_ext INNER JOIN"
    cursor.execute(req, (str(persID),))
    Z = cursor.fetchall()
    for i in range(len(Z)):
        Z[i]=Z[i][0]

    for i in range(len(DBevents)):
        if DBevents[i][0] in Y or DBevents[i][0] in Z:
            persEvents.append(DBevents[i])
```

On **récupère** donc dans la **liste** tous les événements qui concernent l'utilisateur ou un des groupes dont il fait partie en effectuant des **requêtes** en SQL avec la base de données.

Ensuite on va **créer les boutons** et les placer dans une grille avec la fonction « **create_day_boxes** »

```
def create_day_boxes(parent,now):
    global color_bg
    a=70
    current_month = now.month

    current_year = now.year
    nb_jours = calendar.monthrange(current_year,current_month)[1]

    events_of_month1=events_of_month(persEvents,current_month,current_year)

    events = [[] for _ in range(nb_jours)]
    for i in events_of_month1:
        i[1]=i[1].day
        events[i[1]-1]+=i[0]]

    first_day = datetime(current_year, current_month, 1).weekday()

    day_box = Frame(parent,bg=color_bg)

    day_box.grid(row=nb_jours//7, column=nb_jours%7)
    for i in range(nb_jours):
        day_name = datetime.strftime(datetime(current_year, current_month, i+1), '%A')

        if len(events[i])==0:
            color=color_bg
            couleur_rgb = tuple(int(color[i:i+2], 16) for i in (1, 3, 5))
            couleur_hsv = colorsys.rgb_to_hsv(*[c / 255 for c in couleur_rgb])

            couleur_hsv_clair = (couleur_hsv[0], max(0,couleur_hsv[1]*0.4), couleur_hsv[2])
            couleur_rgb_clair = tuple(round(c * 255) for c in colorsys.hsv_to_rgb(*couleur_hsv_clair))
            color = f"#{couleur_rgb_clair[0]:02x}{couleur_rgb_clair[1]:02x}{couleur_rgb_clair[2]:02x}"
        else:
            if a*len(events[i])>240:
                color=' #02x/02x/02x' % (255,0,0)
            else:
                color=' #02x/02x/02x' % (255,240-(a*len(events[i])),0)
            for j in range(len(events[i])):
                if 'vacance' in events[i][j][4] or 'férié' in events[i][j][4]:
                    color=' #02x/02x/02x' % (255,240-(a*len(events[i])),0)
            button = Button(day_box, text="{} {}".format(day_name, i+1),font=('Arial',11, BOLD), activebackground=color,bg,a
            button_text = button.cget("text")
            for j in range(len(events[i])):
                button_text+="\n"+events[i][j][1]
            button.config(text=button_text)
            i+=first_day
            button.grid(row=i//7, column=i%7)
```

Dans une première partie la fonction **récupère uniquement les événements du mois sélectionné** parmi ceux de la liste « persEvents ».

La fonction rentre ensuite dans une **boucle** qui effectue la **même action** pour **chaque jour du mois** :

1. En fonction du nombre d'événement du jour elle associe une couleur à la variable « **color** » qui sera ensuite utilisée comme **couleur du bouton**. Jaune à rouge pour 1 événement jusqu'à 3 et plus, bleu lorsque qu'il s'agit de vacances ou de jours fériés et une couleur pâle s'il n'y a pas d'événements.
2. Le bouton est alors créé avec pour texte **le nom du jour** ainsi que les noms des événements du jour. On lui donne la bonne taille et couleur. La commande « show_day_events » lui est associée afin d'afficher la fenêtre de **détails des événements** lorsqu'on clique sur le bouton.
3. Finalement le bouton est **rangé** dans la grille du mois aux coordonnées correspondant à son jour.

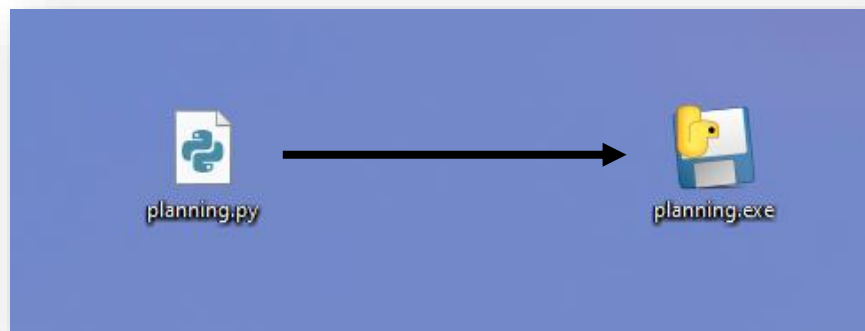
L'ensemble des boutons forment l'**affichage du mois**. Lorsqu'on utilise les boutons pour **changer de mois** il va falloir **refaire cette méthode** pour une date différente. Le planning du mois est **supprimé** et on **réexécute la fonction** « create_day_boxes », les événements récupérés par la fonction seront différents et seront ceux du nouveau mois sélectionné.

Également la **couleur** des boutons utilitaires et du fond de tous les onglets est une **variable générale**. Elle est la même pour tous ces objets ce qui signifie que **la modifier va affecter tous les objets de l'application**. Grâce à cette caractéristique nous avons pu créer un **bouton** à gauche du bouton « refresh » qui fait **changer la couleur de toute l'application instantanément**.

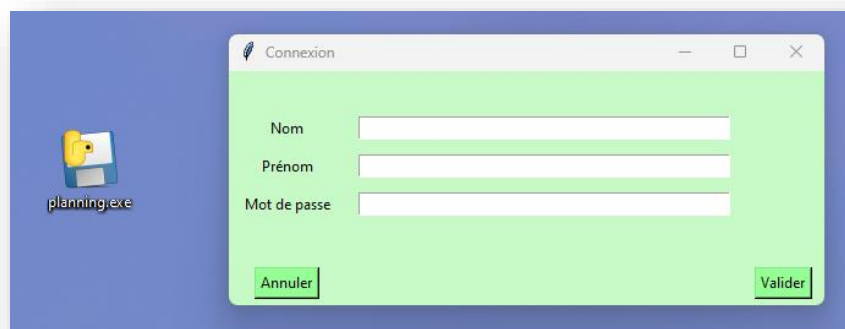


C. CONVERSION EN EXECUTABLE

Lorsque nous avons terminé le **code de l'application**, il faut rendre le code **utilisable** par n'importe qui **sans interpréteur python**. Pour cela on utilise un **script python** qui convertit notre **programme** en une **application exécutable**.



Ainsi il suffit pour **l'utilisateur** d'avoir le **fichier** sur son ordinateur pour lancer **l'application** et se connecter avec ses **identifiants**.



5) DEROULEMENT DU PROJET

Nous avons décidé de travailler sur le projet en se **répartissant les tâches**.

En effet, programmer étant une tâche **assez abstraite** et il est dur de travailler sur la même chose en même temps. En programmant on peut facilement arriver au **même résultat** sans passer par les **mêmes moyens**.

Une fois qu'on a fini notre partie on faisait souvent une petite **réunion** pour savoir jusqu'où on a avancé et ce qu'on doit **faire** pour la **prochaine réunion**.

Nous avons mis en place un horaire le **mercredi après-midi** où nous travaillions sur le projet toutes les semaines.

En travaillant **régulièrement** avec un travail plus approfondi durant certaines périodes comme **les vacances** on arrive à un résultat très **satisfaisant** qui pourra servir à planifier les **événements futurs**.



6) CONCLUSION

En conclusion nous avons réussi à mettre au point une **application utilisable par tout le monde** après l'avoir installée sur son ordinateur. La mise en place du projet se fit principalement en se basant sur le **cahier des charges** et sur les **besoins** de la situation.

Nous avons également dû mettre en place une **organisation** au niveau des horaires, du travail en équipe mais aussi une organisation générale sur **l'avancement du projet** pour continuellement aller dans le même sens et aboutir à une réussite à la fin. Ce protocole ressemble beaucoup à une **démarche d'ingénieur** où il est nécessaire de finir un projet dans les temps tout en respectant les besoins et nécessités de la mission qui lui est attribuée. Nous espérons donc que ce projet nous aura fourni de **l'expérience** utile pour l'avenir.

De plus lors du développement de l'application, nous avons beaucoup appris sur python et sa compatibilité avec SQL. Mettre en place une base de données en ligne était **enrichissant** et nous a appris sur la **gestion des données en ligne**, dans les sites web et les applications notamment.

Finalement nous avons développé nos **compétences sociales** en organisant des réunions de travail et en coopérant tout au long de l'année.

L'application est simple mais **efficace** et elle nous a permis de répondre à de **réelles problématiques** et à nous investir dans un domaine que l'on vise dans le **futur**.