si estas en linux o wsl debes instalar

```
sudo apt install -y python3-venv
```

Poner cada proyecto en su propio ambiente, entrar en cada carpeta.

```
python3 -m venv env

o

python -m venv env
```

Activar el ambiente

```
source env/bin/activate
o:
source env/Scripts/activate
```

verifica lo que tengas instalado .

pip freeze
pasa lo instalado a requeriments.txt

pip freeze > requeriments.txt
pip freeze > requeriments_python3_12.txt

como exportar un env para no saturar disco! :

ejemplo :

primero abrir la terminal , bash , y lo buscas (la carpeta) y la activas :

$ pwd
/d/ORA_proyectos/Python_balances_v02

pero esta en :
C:\Users\gchiappe\Desktop\Ora\BALANCES_GUS_PY
entonces en el bash pones
$ cd ~
$ pwd
/c/Users/gchiappe

$ cd Desktop/Ora/BALANCES_GUS_PY
gchiappe@E102202 MINGW64 ~/Desktop/Ora/BALANCES_GUS_PY (main)

y ahi activas como cualquiera :
source env/Scripts/activate

(env)
gchiappe@E102202 MINGW64 ~/Desktop/Ora/BALANCES_GUS_PY (main)

una vez activado instalas el Jupyter
pip install jupyter

y despues generas el env

python -m ipykernel install --user --name=env --display-name "Python (env)"

una vez terminado , cerras el vscode y lo volves abrir , abris una notebook y
En la parte superior, donde dice "Select Kernel", elige "Python (env)".

Listo

pip install mysql-connector-python
pip install Flask
pip install jsonify
pip install request
pip install flask_cors

from flask import Flask, jsonify , request
import mysql.connector
from flask_cors import CORS
para instalar :

pip install -r requeriments.txt

verificamos en donde esta instalado en este caso con
which pip
which python
which pip3
which python3

instalar dependencia de VSCODE:
vscode-drawio

pip install --force-reinstall -v "Flask-Migrate==3.1.0"

##export FLASK_APP=Flask-Migrate.py

Inicializar el repositorio de migraciones:

flask db init

#Generar una migración:
#flask db migrate -m "Initial migration"

Ver el contenido de la migración:

cat migrations/versions/xxxx_initial_migration.py

Aplicar la migración (opcional):

Si deseas aplicar la migración para sincronizar el estado del modelo con la base de datos, ejecuta:

flask db upgrade

Resumen

SQLAlchemy Directamente: Puedes usar SQLAlchemy para reflejar la metadata de la base de datos y obtener información sobre las tablas y sus columnas.

Flask-Migrate y Alembic: Puedes usar Flask-Migrate con Alembic para generar archivos de migración que reflejan el estado de tu modelo,

lo cual te proporciona una visión general de la estructura de la base de datos.

Dentro de Flask-Migrate, que es una extensión de Flask basada en Alembic para manejar migraciones de bases de datos con SQLAlchemy, existen varios comandos útiles para gestionar cambios en el esquema de la base de datos. Aquí te explico algunos de los comandos más importantes y cuándo conviene usarlos:

1. flask db init

Descripción: Este comando inicializa la carpeta de migraciones en tu proyecto Flask.

Cuándo usarlo:

```
Al comenzar un nuevo proyecto que aún no tiene soporte para migraciones.
Este comando genera una carpeta migrations/ en tu proyecto donde se guardarán
los archivos de migración.
```

2. flask db migrate

Descripción: Genera un nuevo archivo de migración. Este comando compara el esquema actual de la base de datos con los modelos de SQLAlchemy y genera un script de migración para que los cambios en los modelos se apliquen a la base de datos.

Cuándo usarlo:

```
Cuando realizas cambios en tus clases o modelos (por ejemplo, agregas nuevas
tablas, columnas o cambias algún atributo) y quieres que esos cambios se
reflejen en tu base de datos.
```

Nota: Este comando no ejecuta cambios en la base de datos; solo genera el script de migración. Luego debes aplicar los cambios con flask db upgrade.
3. flask db upgrade

Descripción: Aplica los cambios de migración a la base de datos. Si has generado un nuevo script de migración con flask db migrate, este comando se asegura de que esos cambios se apliquen a la base de datos.

Cuándo usarlo:

```
Después de haber generado una migración con flask db migrate.
Cuando necesites actualizar el esquema de la base de datos para que refleje los
cambios recientes en tus modelos de SQLAlchemy.
```

4. flask db downgrade

Descripción: Revierte una o más migraciones, regresando la base de datos a un estado anterior.

Cuándo usarlo:

```
Si cometiste un error en una migración y necesitas deshacer los cambios.
Si quieres regresar a una versión anterior de la base de datos por cualquier
razón.
```

Nota: Este comando no elimina tus migraciones, solo revierte los cambios aplicados a la base de datos.
5. flask db stamp

Descripción: Marca la base de datos en una revisión específica sin aplicar migraciones. Este comando se usa si sabes que tu base de datos ya está sincronizada con una migración determinada, pero por alguna razón la información sobre la versión no está actualizada.

Cuándo usarlo:

```
Si restauras una base de datos desde un backup y necesitas marcar que está en una versión específica de migración, pero no quieres volver a aplicar todas las migraciones.
```

6. flask db current

Descripción: Muestra la versión actual de la base de datos, es decir, la última migración aplicada.

Cuándo usarlo:

```
Cuando quieres verificar qué migración está actualmente aplicada en la base de datos.
```

7. flask db history

Descripción: Muestra un historial de todas las migraciones que se han generado y aplicado.

Cuándo usarlo:

```
Para ver todas las migraciones que has generado a lo largo del tiempo.
Para entender en qué punto del historial de migraciones se encuentra tu base de datos.
```

8. flask db heads

Descripción: Muestra las migraciones "cabeza" o las migraciones más recientes que aún no han sido aplicadas.

Cuándo usarlo:

```
Cuando tienes múltiples ramas de migración y necesitas ver cuál es la cabeza actual
```