1. Definición del DAG:
    - Utiliza VS Code para crear una instancia de la clase DAG.
    - Asigna un identificador único "user_processing" al DAG.
    - Establece la fecha de inicio en el 1 de enero de 2023.
    - Programa el DAG para ejecutarse todos los días a medianoche utilizando un ajuste preestablecido de cron.
    - Establece `catchup=False` para evitar reponer las ejecuciones perdidas.

2. Primera Tarea: Crear Tabla en Postgres
    - Crea tu primera tarea con nombre: "create_table"
    - Utiliza una conexión "postgres" creada en la interfaz de usuario de Airflow.
    - Ejecuta la siguiente consulta SQL:

```
CREATE TABLE IF NOT EXISTS users (
nombre TEXT NOT NULL,
apellido TEXT NOT NULL,
país TEXT NOT NULL,
nombre_usuario TEXT NOT NULL,
contraseña TEXT NOT NULL,
correo_electronico TEXT NOT NULL
);
```

3. Segunda Tarea: Verificar Disponibilidad de la API
    - Crea otra tarea con nombre: "api_available"
    - Crea una conexión HTTP llamada "user_api" con el host: https://randomuser.me/
    - Coloca esta tarea después de la tarea "create_table".

4. Tercera Tarea: Obtener Datos de la API
    - Crea una tarea con nombre: "extract_user"
    - Utiliza la conexión "user_api".
    - Realiza una solicitud GET al punto final "api/".
    - Filtra la respuesta cargándola con `json.loads`.
    - Registra la respuesta en la interfaz de usuario de Airflow.

5. Cuarta Tarea: Procesar Datos Obtendios por extract_user
    - Crea una tarea con nombre: "process_user"
    - Llama a la función `_process_user` que te proporciono en este código. Debes agregarla a tu archivo entre los imports y la definición del DAG.

```
def _process_user(ti):
user = ti.xcom_pull(task_ids="extract_user") # fetch data pushed by the previous task extract_user
user = user['results'][0]
processed_user = json_normalize({
'firstname': user['name']['first'],
'lastname': user['name']['last'],
'country': user['location']['country'],
'username': user['login']['username'],
    'password': user['login']['password'],
    'email': user['email'] })
    processed_user.to_csv('/tmp/processed_user.csv', index=None, header=False)
```

- La función `_process_user` obtiene datos de la tarea `extract_user` usando XCOM.
- Utiliza `json_normalize` para extraer y formatear los datos para almacenarlos en un archivo CSV.
- Establece la dependencia de "extract_user" a "process_user".

6. Quinta Tarea: Almacenar Datos Procesados en Postgres
    - Nombre de la ultima tarea: "store_user"
    - Llama a la función `_store_user`.

```
def _store_user():
hook = PostgresHook(postgres_conn_id='postgres')
hook.copy_expert(
sql="COPY users FROM stdin WITH DELIMITER as ','",
filename='/tmp/processed_user.csv'
)
```

- Que te proporciono aquí y que debes pegar en tu .py después de los imports y antes de la anterior función
- La función utiliza `PostgresHook` para interactuar con Postgres. Utiliza `copy_expert` para exportar datos de un archivo CSV a una tabla de Postgres.
- Establece la dependencia de "process_user" a "store_user".

7. Comprueba que tu flujo se ejecuta correctamente.