# Explicación del Código - Base de Datos de Temperaturas

El código desarrollado tiene como finalidad implementar una base de datos en memoria que permita almacenar, consultar y eliminar registros de temperaturas asociadas a fechas específicas. Cada registro consta de una temperatura (número decimal) y una fecha en formato 'dd/mm/aaaa'. Para garantizar la eficiencia en las operaciones, se utilizó internamente un árbol AVL.

Un árbol AVL es una estructura de datos que mantiene los datos ordenados y balanceados automáticamente. Esto significa que las operaciones como inserción, búsqueda y eliminación se realizan en tiempo logarítmico, incluso cuando el número de datos crece considerablemente. Esta elección es adecuada para el problema planteado, ya que el científico que utiliza el sistema necesita consultar rápidamente datos dentro de un rango de fechas o en una fecha específica.

El sistema está dividido en dos módulos: por un lado, el archivo avl.py implementa toda la lógica del árbol AVL, incluyendo la estructura de los nodos, los métodos para insertar, eliminar, buscar y recorrer el árbol. Cada nodo del árbol almacena como clave una fecha transformada al tipo datetime.date, lo cual permite comparar y ordenar correctamente las fechas. Además, cada nodo guarda el valor de la temperatura correspondiente a esa fecha.

Por otro lado, el archivo temperaturas_db.py contiene la clase Temperaturas_DB, que representa la interfaz que utilizará el usuario para trabajar con la base de datos. Esta clase encapsula el árbol AVL y ofrece métodos accesibles como guardar_temperatura, devolver_temperatura, borrar_temperatura, max_temp_rango, min_temp_rango, temp_extremos_rango, devolver_temperaturas y cantidad_muestras. Todos estos métodos convierten las fechas ingresadas por el usuario a objetos datetime antes de interactuar con el árbol, lo que garantiza la validez y consistencia de los datos.

La elección de usar un árbol AVL y encapsular la lógica en clases separadas responde a buenas prácticas de programación: permite mantener el código modular, reutilizable y fácil de extender o mantener. Además, se agregaron validaciones y manejo de excepciones para garantizar que los errores se detecten de manera clara y se informe al usuario si intenta consultar o eliminar una fecha que no existe, o si el formato de fecha ingresado no es válido.

En resumen, la solución implementada es eficiente y robusta, utilizando estructuras de datos avanzadas de manera justificada y manteniendo una organización clara del código. Esto asegura que el sistema pueda crecer en volumen de datos sin perder rendimiento y que sea fácil de usar y mantener en contextos reales.