PROGRAMACIÓN II

1º AÑO

Clase Nº 8: Bases de Datos con SQLAlchemy y Flask

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Introducción a las bases de datos relacionales.
- Conexión de una aplicación Flask con una base de datos.
- ORM con SQLAlchemy: definición de modelos.
- Operaciones básicas: crear, leer, actualizar y eliminar (CRUD).
- Consultas simples y avanzadas usando SQLAlchemy.
- Actividades prácticas integradas al desarrollo.

1. Presentación

En la clase anterior dimos nuestros primeros pasos en la creación de formularios dinámicos con Flask, aprendiendo a construir interfaces donde los usuarios pueden interactuar con nuestras aplicaciones web. Descubrimos cómo recibir y procesar datos del lado del servidor, utilizando HTML y rutas de Flask para generar contenido personalizado.

Sin embargo, notamos una limitación importante: toda esa información desaparecía una vez que cerrábamos la aplicación o reiniciábamos el servidor.

Hoy vamos a superar ese obstáculo incorporando un nuevo componente esencial en el desarrollo web: **las bases de datos relacionales**.

Este avance marca un antes y un después en nuestro recorrido. Aprenderemos a almacenar información de manera **persistente**, es decir, que se conserva aunque apaguemos la computadora o reiniciemos el servidor. A partir de ahora, podremos construir sistemas que guarden datos reales como usuarios, productos, comentarios o estadísticas, y que puedan consultarse y modificarse cuando sea necesario.

Para lograr esto, utilizaremos **SQLAlchemy**, una herramienta fundamental dentro del ecosistema Flask. SQLAlchemy es un ORM (Object-Relational Mapper), lo que significa que nos permite trabajar con bases de datos desde Python, usando clases y objetos en lugar de escribir consultas SQL manualmente. Esto no solo hace que el código sea más legible y seguro, sino que también permite modelar estructuras complejas de datos de manera clara y ordenada.

Así, daremos un paso clave en la construcción de aplicaciones modernas, escalables y preparadas para gestionar grandes volúmenes de información. Lo que hasta ahora eran prácticas aisladas, comenzarán a integrarse en proyectos más robustos y cercanos al mundo profesional.

Actividad 1: Mira el siguiente video y reflexionar sobre el CRUD con SQLALCHEMY:

CRUD PYTHON + BASE DE DATOS + SQLALCHEMY.

El video nos ofrece una guía práctica que muestra cómo implementar las operaciones básicas (crear, leer, actualizar y eliminar) usando SQLAlchemy en Python.

A continuación, daremos los primeros pasos prácticos para configurar SQLAlchemy, definir nuestros modelos de datos y empezar a interactuar con la base de datos desde Python.

1. Desarrollo

¿Qué es una base de datos relacional?

Una base de datos relacional es un sistema que permite almacenar, organizar y consultar información de forma estructurada. Su modelo se basa en tablas, similares a las de una hoja de cálculo, que contienen filas (también llamadas registros) y columnas (también llamadas campos). Cada tabla representa una entidad concreta del sistema, como, por ejemplo:

Una tabla usuarios, con columnas como id, nombre, email, etc.

Una tabla productos, con columnas como id, nombre, precio, stock, etc.

Las bases de datos relacionales permiten también definir relaciones entre estas tablas. Por ejemplo:

- Un usuario puede realizar muchos pedidos (relación uno a muchos).
- Un producto puede pertenecer a una categoría (relación muchos a uno).

- Un alumno puede inscribirse en varios cursos y un curso puede tener varios alumnos (relación muchos a muchos).

Estas relaciones se definen utilizando claves primarias (primary keys) y claves foráneas (foreign keys), y son fundamentales para modelar información compleja y conectada.

Tipos de motores de bases de datos relacionales

Existen muchos motores o sistemas de gestión de bases de datos relacionales (RDBMS), entre los más conocidos se encuentran:

- SQLite: Es una base de datos embebida que se guarda en un solo archivo del sistema. No requiere instalación de un servidor y es muy útil para desarrollar y testear aplicaciones pequeñas o medianas de forma rápida. Su simplicidad la hace ideal para ambientes de aprendizaje, prototipado y aplicaciones de escritorio.

- MySQL: Es uno de los motores más populares y utilizados en entornos de producción. Permite múltiples conexiones simultáneas, tiene alto rendimiento y soporta grandes volúmenes de datos. A diferencia de SQLite, necesita configurarse como un servicio de base de datos, generalmente en un servidor (local o remoto).

En esta clase utilizaremos SQLite por su facilidad de uso, pero los conocimientos

adquiridos serán completamente aplicables a sistemas como MySQL o PostgreSQL.

¿Qué es SQLAlchemy y por qué lo usamos?

SQLAlchemy es una biblioteca muy poderosa de Python que permite interactuar con bases de datos relacionales de forma eficiente, segura y elegante.

En lugar de escribir directamente comandos en SQL como:

```sql
SELECT * FROM usuarios WHERE email = 'ejemplo@correo.com';
```

Podemos utilizar código Python más legible y orientado a objetos:

```python
Usuario.query.filter_by(email='ejemplo@correo.com').first()
```

Esto se debe a que SQLAlchemy es un **ORM** (*Object-Relational Mapper*), es decir, un sistema que **traduce automáticamente las clases y objetos de Python en estructuras equivalentes en la base de datos**. Cada clase

representa una tabla, y cada instancia de esa clase representa una fila o registro.

Ventajas de usar SQLAlchemy	
Abstracción	No es necesario escribir consultas SQL manualmente para operaciones básicas.
Reutilización de código	Usamos clases y métodos de Python para representar y manipular datos.
Seguridad	Previene errores comunes como las inyecciones SQL.
Compatibilidad	Funciona con diferentes motores de base de datos (SQLite, MySQL, PostgreSQL, etc.)
Legibilidad y mantenibilidad	El código es más claro, fácil de leer y de mantener a largo plazo.

Instalación de SQLAlchemy

Antes de empezar a trabajar con bases de datos, debemos asegurarnos de que Flask y SQLAlchemy estén instalados. SQLAlchemy es una biblioteca que se instala por separado.

Para instalarla, usamos pip en la terminal:

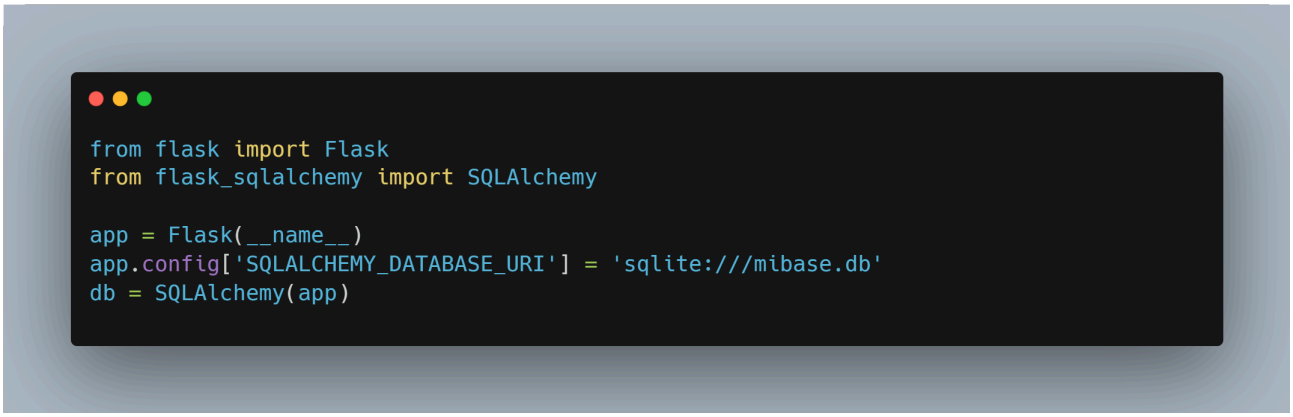
```
pip install flask_sqlalchemy
```

Este comando descarga e instala la extensión Flask-SQLAlchemy, que permite integrar el ORM con una aplicación Flask.

Configuración en Flask

Para usar SQLAlchemy en Flask, debemos agregar algunas líneas en nuestro archivo `app.py`. Estas permiten definir la base de datos y conectar SQLAlchemy con la app.

Ejemplo básico usando SQLite:



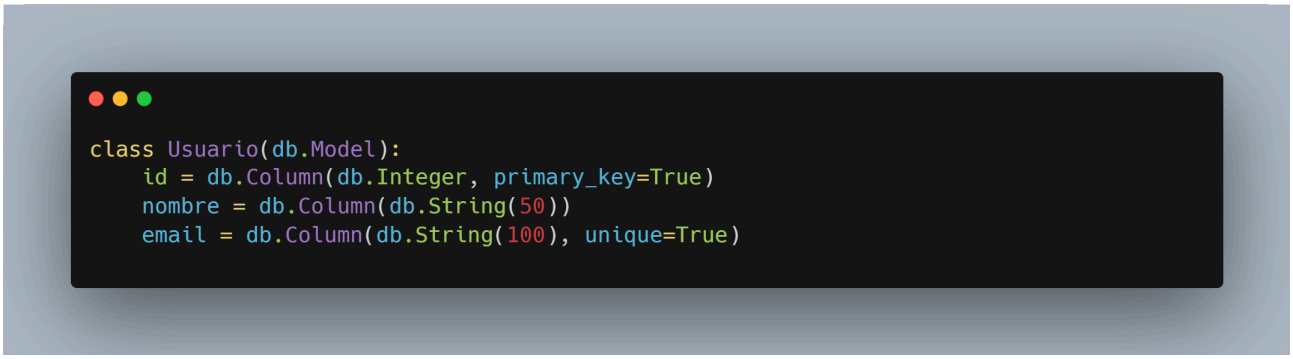
```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///mibase.db'
db = SQLAlchemy(app)
```

- *SQLALCHEMY_DATABASE_URI*: indica la dirección de la base de datos. En este caso, usaremos SQLite y se generará un archivo llamado `mibase.db`.
- *db = SQLAlchemy(app)*: crea una instancia del ORM y la asocia a nuestra aplicación Flask.

Definición de modelos

Un **modelo** representa una **tabla** en la base de datos. Cada clase que definimos hereda de “db.Model” y sus atributos representan columnas.

Ejemplo:



```python
class Usuario(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(50))
    email = db.Column(db.String(100), unique=True)
```

- *id*: es una columna de tipo entero, clave primaria, se genera automáticamente.
- *nombre*: es un texto de hasta 50 caracteres, obligatorio (nullable=False).
- *email*: es un texto único (no se repite en otros registros) y obligatorio.

Crear la base de datos

Una vez definidos los modelos, podemos crear las tablas en el archivo de base de datos con:

```python
db.create_all()
```

Esto se ejecuta una única vez, justo después de definir los modelos.

Actividad 2: Crear tu primera base de datos

Objetivo: Implementar tu primer modelo y base de datos funcional usando SQLAlchemy.

- Crea un archivo `app.py` e inicializá Flask y SQLAlchemy.

- Definí una clase `Producto` con los campos: id, nombre, precio.
- Usá `db.create_all()` para crear la tabla.
- Insertá 3 productos utilizando `db.session.add()`.
- Consultá los productos con `Producto.query.all()` y mostralos por consola.

Operaciones CRUD con SQLAlchemy

El término **CRUD** representa las cuatro operaciones básicas que se pueden realizar sobre los datos en una base de datos:

- **Create (Crear)**: insertar nuevos registros.
- **Read (Leer)**: consultar o recuperar información almacenada.
- **Update (Actualizar)**: modificar datos existentes.
- **Delete (Eliminar)**: borrar registros de forma permanente.

Estas operaciones son fundamentales en cualquier aplicación web que gestione información dinámica, como usuarios, productos, publicaciones, etc.

Como **SQLAlchemy**, funciona como un ORM, podemos realizar todas estas acciones utilizando clases y objetos en lugar de escribir directamente sentencias SQL.

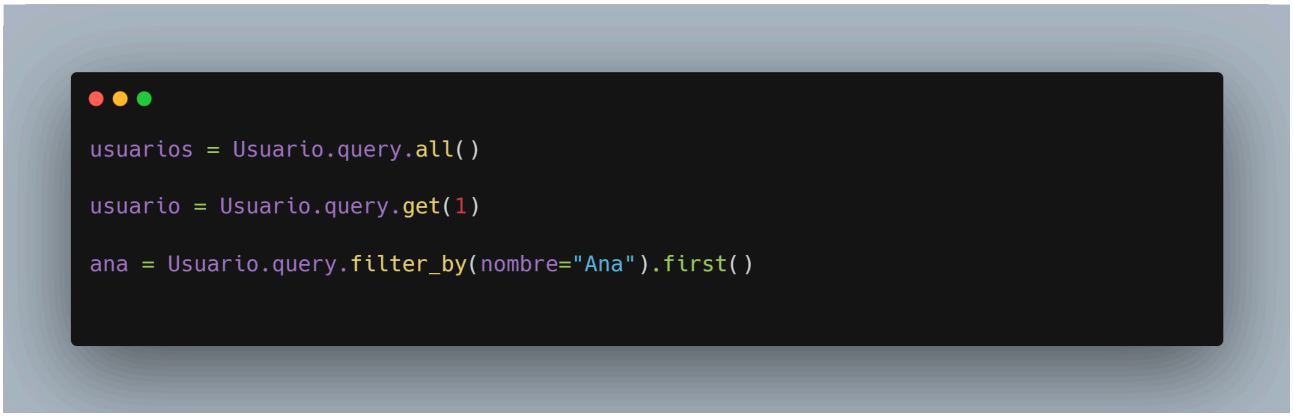
Veamos cómo implementar cada una de estas operaciones usando SQLAlchemy en una aplicación Flask:

✅ CREATE – Crear un nuevo registro

```python
nuevo_usuario = Usuario(nombre="Ana", email="ana@mail.com")

db.session.add(nuevo_usuario)

db.session.commit()
```

🔎 Creamos una instancia del modelo Usuario con los datos que queremos guardar. Luego, la agregamos a la sesión de la base de datos y finalmente confirmamos la operación con commit() para que se almacene.

🔍 READ – Leer o consultar registros



```python
usuarios = Usuario.query.all()

usuario = Usuario.query.get(1)

ana = Usuario.query.filter_by(nombre="Ana").first()
```

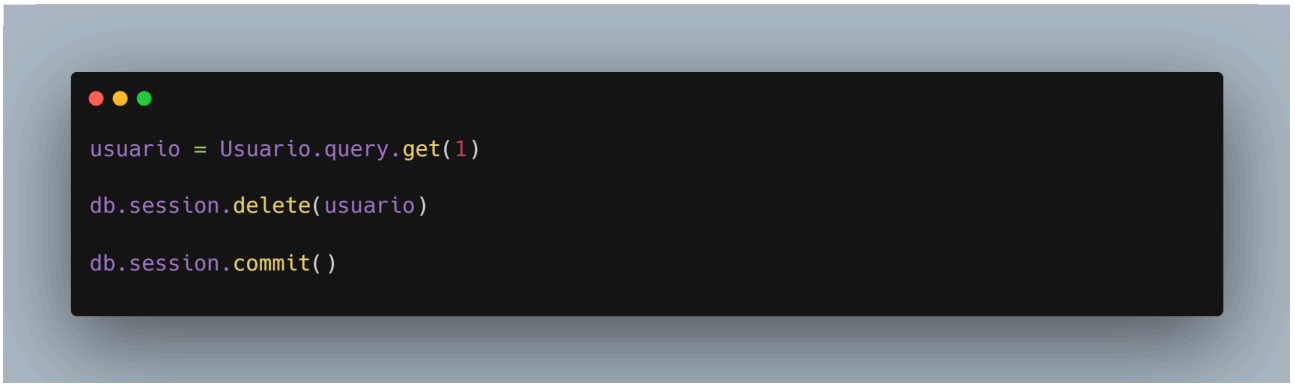
🔎 Con ".query.all()" obtenemos todos los registros. Con ".get(id)" accedemos por clave primaria. Con ".filter_by()" podemos buscar usando condiciones específicas (como una cláusula WHERE de SQL).

✏️ UPDATE – Actualizar un registro existente

```python
usuario = Usuario.query.get(1)

usuario.email = "nuevo_correo@mail.com"

db.session.commit()
```

🔎 Primero buscamos el objeto, luego cambiamos sus atributos y finalmente hacemos commit() para que los cambios se reflejen en la base de datos.

❌ DELETE – Eliminar un registro



```python
usuario = Usuario.query.get(1)

db.session.delete(usuario)

db.session.commit()
```

🔎Al igual que en los casos anteriores, se busca el objeto, se elimina con delete() y se confirman los cambios con commit().

Actividad 3: Primer CRUD con SQLAlchemy

Objetivo: Crear un modelo de base de datos y realizar operaciones básicas.

- Crear un modelo llamado Producto con los campos: id, nombre, precio.
- Insertar al menos tres productos.

- Consultar y mostrar los productos en consola.
- Actualizar uno de ellos y eliminar otro.
- Explicar en una línea para qué sirve `db.session.commit()`.

Consultar registros en SQLAlchemy

Una de las funcionalidades más importantes al trabajar con bases de datos es poder **consultar información** de manera flexible. SQLAlchemy nos permite hacer esto de forma muy intuitiva, utilizando métodos como query.all(), filter(), filter_by() o order_by() directamente sobre los modelos definidos.

Actividad 4: Consultar registros con ayuda de ChatGPT

Instrucciones:

Abrí **ChatGPT** y escribí este prompt:

> "Pdele a ChatGPT una versión optimizada de tu consulta. Compará ambas versiones y justificá cuál preferís.""

Leé la respuesta de ChatGPT y copiá solo la línea o bloque de código que hace la consulta.

Pegalo en tu proyecto y probalo con tu modelo Producto.

Respondé:

> ¿La consulta funcionó?
>
> ¿Cambiarías algo del ejemplo que te dio?

A continuación, se presentan distintos ejemplos prácticos para consultar

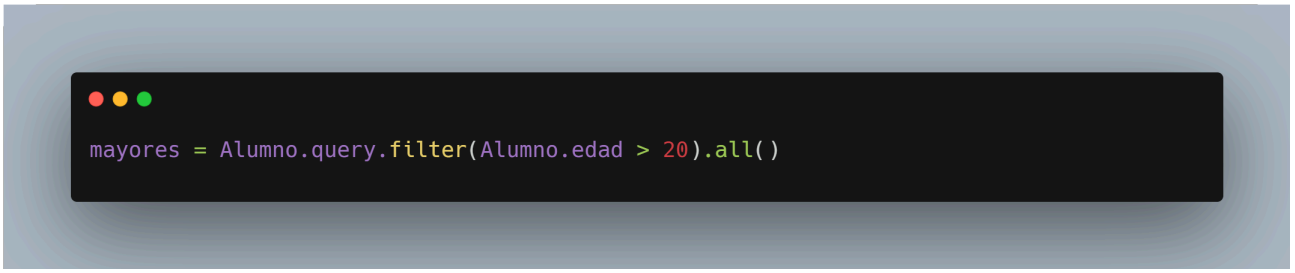
registros en una tabla, aplicando filtros, ordenamientos, límites y conteos, sin necesidad de escribir sentencias SQL manuales.

```
alumnos = Alumno.query.all()
```

Este método devuelve una lista con todos los registros de la tabla `Alumno`. Es útil para mostrar todos los datos disponibles.

```
alumno = Alumno.query.get(1)
```

Busca un registro por su clave primaria (en este caso, `id=1`). Si no lo encuentra, devuelve `None`. Ideal para acceder a un objeto específico.



```
mayores = Alumno.query.filter(Alumno.edad > 20).all()
```

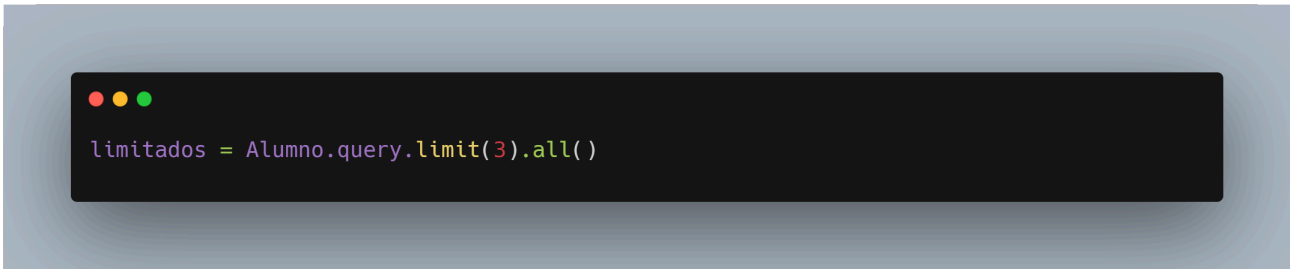
Devuelve todos los alumnos cuya edad es mayor a 20. La función `filter` permite aplicar condiciones utilizando operadores lógicos (`>`, `<`, `==`, `!=`).

```
primero = Alumno.query.filter_by(nombre="Laura").first()
```

`filter_by` permite buscar usando nombre de columnas como argumentos. `first()` retorna el primer resultado encontrado o `None`.

```
ordenados = Alumno.query.order_by(Alumno.nombre.asc()).all()
```

Ordena los resultados alfabéticamente por nombre (de forma ascendente). También puede usarse `.desc()` para descendente.



```
limitados = Alumno.query.limit(3).all()
```

Limita la cantidad de registros devueltos a los primeros 3. Muy útil para paginación o vistas reducidas.

```python
conteo = Alumno.query.count()
```

Devuelve la cantidad total de registros en la tabla. Ideal para mostrar estadísticas.

```python
existe = Alumno.query.filter_by(nombre="Ana").first() is not None
```

Verifica si existe al menos un alumno con nombre Ana. Útil para validaciones previas.

Instancia de Autoevaluación (OBLIGATORIO): SQLAlchemy y Flask

📌 **Objetivo de la actividad:** Para reforzar los conocimientos adquiridos en la clase a través de una autoevaluación individual que permita a cada estudiante verificar su comprensión sobre el uso de bases de datos con SQLAlchemy dentro de una aplicación Flask.

📋**Modalidad:** Cuestionario de autoevaluación en la plataforma virtual (Moodle), con retroalimentación automática.

Criterios de evaluación del cuestionario:

- Comprensión de los conceptos CRUD.
- Capacidad de interpretar y ejecutar consultas SQLAlchemy.
- Aplicación práctica en un caso Flask.

ACTIVIDAD INTEGRADORA DE CIERRE

En esta actividad vas a aplicar todos los conocimientos trabajados en la clase: la creación de una app con Flask, la conexión con una base de datos usando SQLAlchemy, la definición de modelos, y las operaciones básicas para guardar y mostrar datos.

Consigna:

1. Creá un archivo app.py.
2. Armá una aplicación con Flask conectada a una base de datos SQLite con SQLAlchemy.
3. Define un modelo llamado Producto que tenga:
    - id (clave primaria)
    - nombre
    - recio
4. Cargá desde el código al menos 3 productos.
5. Creá una ruta (por ejemplo /productos) que muestre los productos guardados.

Cierre:

Hoy dimos un paso clave en nuestro camino como futuros profesionales en Ciencia de Datos e Inteligencia Artificial: incorporamos el uso de bases de datos dentro de nuestras aplicaciones en Python, utilizando Flask y SQLAlchemy como herramientas de conexión entre el mundo del código y el almacenamiento estructurado de información.

Aprendimos a definir modelos que representan entidades reales, a manipular datos con operaciones CRUD, y a mostrar esa información de forma dinámica utilizando HTML y Jinja2. Estos conceptos no solo son importantes para el desarrollo web, sino también para cualquier sistema que necesite almacenar, consultar y procesar información: desde sistemas de gestión hasta aplicaciones de análisis de datos.

Lo más importante es entender que la programación es una herramienta transversal. No es un fin en sí mismo, sino un medio que potencia nuestra capacidad de construir soluciones, automatizar procesos, visualizar información y aplicar modelos inteligentes.

➡️ En la próxima clase, vamos a integrar todo lo aprendido y llevarlo un paso más allá: construiremos una aplicación completa que podrá ejecutarse como un programa independiente, usando PyInstaller. Esto nos permitirá distribuir nuestras aplicaciones como ejecutables, sin necesidad de instalar Python, lo que facilita su uso en entornos reales.

Será una experiencia muy cercana al desarrollo profesional: crear,

empaquetar y ejecutar una aplicación real.

🚀 Seguimos avanzando. Cada línea de código que escribimos, cada proyecto que construimos, nos acerca más a aplicar la inteligencia artificial y la ciencia de datos con impacto en el mundo real.

Bibliografía obligatoria

- Ramírez Jiménez, Ó. (2021). *Python a fondo: Domina el lenguaje, la programación orientada a objetos y los aspectos avanzados*. Marcombo.

- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

Recursos adicionales recomendados

1. **Documentación oficial de SQLAlchemy (en inglés):**
   https://docs.sqlalchemy.org/en/20/
   Guía completa del ORM, incluye ejemplos, modelos, consultas, relaciones y más.
2. **Flask-SQLAlchemy – Extensión oficial para integrar SQLAlchemy con Flask:**
   https://flask-sqlalchemy.palletsprojects.com/
   Instrucciones de instalación, configuración, uso de modelos y consultas dentro de proyectos Flask.
3. **Introducción práctica a bases de datos relacionales y ORM en Python – Programación en Python:**
   https://ellibrodepython.com/sqlalchemy
   Explicaciones en español y ejemplos paso a paso sobre cómo usar SQLAlchemy.
4. **Video recomendado: CRUD Python + SQLAlchemy (YouTube):**
   https://www.youtube.com/watch?v=ZsQYqNbBevE

Muestra práctica del uso de SQLAlchemy para realizar operaciones CRUD completas en Python.

5. **Tutorial interactivo sobre POO y ORM (en español – DataCamp):**
   https://www.datacamp.com/es/tutorial/python-oop-tutorial
   Aunque se centra en POO, es un buen complemento para entender cómo funciona SQLAlchemy internamente.