«Talento Tech»

Iniciación a la

Programación con Python

Clase 15

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 15 | Trabajo Final Integrador y repaso general

Temario:

- Integración de todos los conceptos aprendidos.
- Desarrollo de un proyecto completo.
- Mejores prácticas de programación.
- Documentación y presentación del proyecto.

Objetivos de la Clase

En esta última clase del curso, integrarás todo lo aprendido en un proyecto funcional que servirá como modelo para tu **Trabajo Final Integrador (TFI)**. Aplicarás los conocimientos adquiridos sobre **bases de datos SQLite3, funciones en Python, manejo de excepciones, validaciones, seguridad, estructuras de datos y optimización del código**.

A través del desarrollo de un **sistema CRUD completo para la gestión de alumnos**, consolidarás tus habilidades en la manipulación de datos, creando, consultando, actualizando y eliminando información de una base de datos real. Además, reforzarás el uso de **buenas prácticas de programación**, asegurando que el código sea modular, seguro y eficiente.

También aprenderás a **documentar correctamente tu código** para facilitar su comprensión y mantenimiento. Por último, te prepararemos para la **presentación profesional de tu proyecto**, explorando estrategias para estructurar y explicar tu solución de manera clara y efectiva.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

El desafío final en TalentoLab 🧩

Talento Lab

El ambiente en **TechLab** se siente distinto hoy. Después de semanas de aprendizaje y práctica, Mariana reúne a todo el equipo para plantear el último desafío antes de la entrega del proyecto final. Con una sonrisa, toma la palabra.

"—Llegó el momento de demostrar todo lo que aprendiste. Has trabajado con **bases de datos, manejo de errores, validaciones, funciones, consultas SQL y muchas otras herramientas fundamentales**. Ahora, el cliente necesita una solución completa, bien estructurada y funcional."

Diego, el desarrollador senior, interviene:

"— La mejor forma de prepararte para el **Trabajo Final Integrador** es construir un **sistema CRUD** desde cero. Te propongo un desafío muy parecido al proyecto real, pero con una temática distinta. Vamos a desarrollar un sistema de **gestión de alumnos** que permita registrar, consultar, actualizar y eliminar información en una base de datos."

Mariana asiente y agrega:

"— Este ejercicio te ayudará a consolidar lo aprendido, te servirá como base para organizar tu propio proyecto final. Quiero que prestes atención a la estructura del código, las validaciones y la documentación. Todo lo que hagamos hoy será clave para entregar un trabajo profesional."

Diego finaliza la reunión con un consejo:

"— Te recomiendo que dividas el desarrollo en partes: primero, la **estructura de la base de datos**, luego las **funciones CRUD**, y finalmente la **interfaz de interacción con el usuario**. Aplica lo que hemos visto sobre **buenas prácticas, modularidad y seguridad**."

Creación de la base de datos y estructura de tablas

Antes de comenzar a programar, es fundamental definir cómo vamos a almacenar los datos de los alumnos en nuestra base de datos. Usaremos **SQLite3**, que nos permitirá gestionar la información de manera estructurada y eficiente dentro de nuestro programa.

Para esta aplicación, crearemos una base de datos llamada **instituto.db**, que almacenará la información de los alumnos. La base de datos contará con una tabla principal llamada **alumnos**, con los siguientes campos:

Campo	Tipo de dato	Descripción
id	INTEGER PRIMARY KEY AUTOINCREMENT	Identificador único del alumno.
nombre	TEXT NOT NULL	Nombre del alumno. No puede estar vacío.
apellido	TEXT NOT NULL	Apellido del alumno. No puede estar vacío.
edad	INTEGER CHECK (edad >= 5 AND edad <= 120)	Edad del alumno, con un rango de validación.
curso	TEXT NOT NULL	Curso en el que está inscripto.
email	TEXT UNIQUE CHECK (email LIKE '%@%')	Dirección de correo electrónico válida y única.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Cada campo cumple un propósito específico. Usamos una **clave primaria autoincremental** para el **id**, asegurándonos de que cada alumno tenga un identificador único. Aplicamos restricciones como **NOT NULL** en campos que siempre deben tener un valor y validaciones como **CHECK** en **edad** para asegurarnos de que sea un número válido. También nos aseguramos de que el **email** tenga el formato correcto.

Ahora que definimos la estructura, vamos a crear la base de datos y la tabla utilizando Python y SQLite3. Este código primero establece una conexión con **instituto.db** y luego crea la tabla **alumnos** con las restricciones que definimos previamente. La línea **IF NOT EXISTS** en la sentencia **CREATE TABLE** evita errores si la tabla ya existe.

```
import sqlite3
from colorama import Fore, Style, init
# Inicializar colorama
init(autoreset=True)

# Conectar o crear la base de datos
conexion = sqlite3.connect("instituto.db")
cursor = conexion.cursor()

# Crear la tabla de alumnos
cursor.execute('''
   CREATE TABLE IF NOT EXISTS alumnos (
       id INTEGER PRIMARY KEY AUTOINCREMENT,
       nombre TEXT NOT NULL,
       apellido TEXT NOT NULL,
       edad INTEGER CHECK (edad >= 5 AND edad <= 120),
       curso TEXT NOT NULL,
       email TEXT UNIQUE CHECK (email LIKE '%@%')
   )
''')

# Confirmar los cambios y cerrar la conexión
conexion.commit()
conexion.close()

print("Base de datos y tabla de alumnos creadas con éxito.")
```

El código establece una conexión con **SQLite3** y creando un objeto **cursor**, que nos permite ejecutar comandos SQL dentro de la base de datos. Luego, con **CREATE TABLE IF NOT EXISTS alumnos**, definimos nuestra tabla asegurándonos de que solo se cree si no existe previamente.

Cada campo de la tabla tiene restricciones para garantizar la calidad de los datos. Por ejemplo, **NOT NULL** evita que se inserten registros incompletos, **CHECK** impide edades fuera del rango definido, y **UNIQUE** en email asegura que no haya duplicados.

Finalmente, **conexion.commit()** guarda los cambios en la base de datos y **conexion.close()** cierra la conexión para liberar recursos.

Función para agregar alumnos a la base de datos

Ahora que tenemos nuestra base de datos y la tabla **alumnos** creada, el siguiente paso es implementar la función que permitirá registrar nuevos alumnos. Esta función debe permitir la entrada de datos del usuario, validarlos y almacenarlos en la base de datos de forma segura.

Para ello, agregaremos validaciones en la entrada de datos y utilizaremos **try-except** para manejar posibles errores, como intentos de insertar datos duplicados o en un formato incorrecto.

```
def agregar_alumno():
    """
    Solicita los datos del alumno y los guarda en la base de datos,
validando la entrada.
    """

    # Conectar a la base de datos
    conexion = sqlite3.connect("instituto.db")
    cursor = conexion.cursor()

    try:
        print(Fore.CYAN + "\n=== Registrar Nuevo Alumno ===")
```

```
        # Solicitar datos con validaciones básicas
        nombre = input("Ingresá el nombre del alumno:
").strip().capitalize()
        apellido = input("Ingresá el apellido del alumno:
").strip().capitalize()
        edad = input("Ingresá la edad del alumno: ").strip()
        curso = input("Ingresá el curso del alumno: ").strip().upper()
        email = input("Ingresá el email del alumno: ").strip().lower()

        # Validaciones
        if not nombre or not apellido or not curso:
            print(Fore.RED + "[ERROR] El nombre, apellido y curso no
pueden estar vacíos.")
            return

        if not edad.isdigit() or not (5 <= int(edad) <= 120):
            print(Fore.RED + "[ERROR] La edad debe ser un número entre 5
y 120.")
            return

        if "@" not in email or " " in email:
            print(Fore.RED + "[ERROR] El email debe tener un formato
válido (ejemplo@correo.com).")
            return

        # Insertar en la base de datos
        cursor.execute('''
            INSERT INTO alumnos (nombre, apellido, edad, curso, email)
            VALUES (?, ?, ?, ?, ?)
        ''', (nombre, apellido, int(edad), curso, email))

        # Confirmar la transacción
        conexion.commit()
        print(Fore.GREEN + f"[ÉXITO] Alumno {nombre} {apellido}
registrado correctamente.")
```

```
    except sqlite3.IntegrityError:
        print(Fore.RED + "[ERROR] El email ingresado ya está
registrado.")

    except sqlite3.Error as e:
        print(Fore.RED + f"[ERROR] Error en la base de datos: {e}")

    finally:
        conexion.close()
```

El código que acabás de ver permite registrar alumnos en la base de datos, asegurando que los datos sean válidos y manejando posibles errores. Vamos a recorrerlo en detalle para que puedas entender cada parte y cómo se conecta con lo que ya aprendiste.

El primer paso es la importación de los módulos necesarios. Se usa **sqlite3** para interactuar con la base de datos y **Colorama** para mejorar la visualización de mensajes en la terminal, destacando los errores en rojo, mostrar información en cian y confirmar operaciones exitosas en verde.

Luego, se define la función **agregar_alumno()**, que se encarga de solicitar los datos al usuario, validarlos e insertarlos en la base de datos. Lo primero que hace esta función es conectarse a la base de datos **instituto.db** y obtener un cursor para ejecutar comandos SQL.

Antes de realizar la inserción en la base de datos, el programa solicita al usuario cinco datos: nombre, apellido, edad, curso y email. Se aplican transformaciones básicas en las cadenas de texto, como convertir el nombre y el apellido a **formato capitalizado**, asegurarse de que el curso se almacene en **mayúsculas** y que el email se almacene en **minúsculas**. Esto ayuda a mantener los datos consistentes dentro de la base.

Una vez obtenidos los datos, el código realiza varias validaciones antes de insertarlos. Primero, verifica que ni el nombre, ni el apellido, ni el curso estén vacíos, ya que son datos obligatorios. Si alguno está vacío, se muestra un mensaje de error y se corta la ejecución con un **return**. Luego, revisa que la edad sea un número entero entre 5 y 120 años, ya que no tendría sentido registrar un alumno con una edad fuera de esos rangos. También valida que el email contenga un **@** y que no tenga espacios en blanco, asegurando que el formato sea correcto. Si alguna de estas validaciones falla, el programa no intenta guardar los datos y muestra un mensaje de error al usuario.

Cuando todos los datos son correctos, se ejecuta la consulta **SQL INSERT INTO alumnos** para agregar el nuevo registro a la base de datos. Acá es importante notar que se usa una consulta parametrizada, representando los valores con signos de interrogación **(?)**. Esto es clave porque previene ataques de inyección SQL y asegura que los datos sean insertados de manera segura.

Después de ejecutar la consulta, se usa **commit()** para confirmar los cambios y guardar el nuevo registro en la base de datos. Si todo sale bien, se muestra un mensaje en verde confirmando que el alumno fue agregado con éxito.

En el bloque **except**, se manejan posibles errores que pueden surgir durante la ejecución. Si se intenta registrar un email que ya está en uso, el programa detecta la violación de una restricción de unicidad y muestra un mensaje de error específico. Si ocurre otro problema con SQLite, el mensaje de error de la base de datos se captura y se muestra al usuario, para que pueda entender qué sucedió.

Por último, en el bloque **finally**, el código **se asegura de cerrar la conexión con la base de datos**, independientemente de si la operación fue exitosa o falló. Esto es importante porque mantener conexiones abiertas innecesariamente puede generar problemas en el sistema.

Consultar y modificar datos

Como ya sabemos, en cualquier sistema de gestión es tan importante registrar información como poder acceder a ella y modificarla cuando sea necesario. Vamos a trabajar en dos funciones clave: una para **consultar alumnos** y otra para **eliminarlos de la base de datos.**

Comenzaremos con la consulta. Esta funcionalidad permitirá ver los datos almacenados y realizar búsquedas específicas. Implementaremos una opción que muestre **todos los alumnos** y otra que permita **filtrar por nombre**. Además, aplicaremos formato con Colorama para mejorar la experiencia visual.

Luego, pasaremos a la eliminación de alumnos. Antes de eliminar un registro, es fundamental asegurarnos de que el usuario realmente desea hacerlo. Para eso, pediremos una **confirmación** antes de ejecutar la acción. Además, como los alumnos pueden estar relacionados con otras tablas en la base de datos, utilizaremos **transacciones** para garantizar que cualquier error en la operación no deje la base en un estado inconsistente.

Vamos a implementar estas funciones asegurándonos de utilizar:

- **Consultas parametrizadas** para evitar inyecciones SQL.
- **Manejo de errores** con try-except para capturar problemas como intentos de eliminar un alumno inexistente.
- **Colorama** para destacar la información y mejorar la legibilidad de la terminal.

Comencemos por la función **consultar_alumno()**, que nos permite listar todos los alumnos o buscar por nombre.

```
def consultar_alumnos():
    """
    Permite consultar la lista de alumnos registrados en la base de
datos.
    Ofrece la opción de ver todos los alumnos o buscar por nombre.
    """

    # Conectar a la base de datos
    conexion = sqlite3.connect("instituto.db")
    cursor = conexion.cursor()

    try:
        print(Fore.CYAN + "\n=== Consultar Alumnos ===")
        print("1. Mostrar todos los alumnos")
        print("2. Buscar alumno por nombre")
        opcion = input("Seleccioná una opción: ").strip()

        if opcion == "1":
            # Mostrar todos los alumnos
            cursor.execute("SELECT id, nombre, apellido, edad, curso,
email FROM alumnos")
            alumnos = cursor.fetchall()

            if not alumnos:
                print(Fore.YELLOW + "[INFO] No hay alumnos registrados
en la base de datos.")
                return
```

```
            print(Fore.GREEN + "\n=== Lista de Alumnos ===")
            for alumno in alumnos:
                print(Fore.WHITE + f"ID: {alumno[0]}, Nombre:
{alumno[1]} {alumno[2]}, Edad: {alumno[3]}, Curso: {alumno[4]}, Email:
{alumno[5]}")

        elif opcion == "2":
            # Buscar alumno por nombre
            nombre_busqueda = input("Ingresá el nombre del alumno a
buscar: ").strip().capitalize()

            if not nombre_busqueda:
                print(Fore.RED + "[ERROR] El nombre no puede estar
vacío.")
                return

            cursor.execute("SELECT id, nombre, apellido, edad, curso,
email FROM alumnos WHERE nombre = ?", (nombre_busqueda,))
            alumnos = cursor.fetchall()

            if not alumnos:
                print(Fore.YELLOW + "[INFO] No se encontraron alumnos
con ese nombre.")
                return

            print(Fore.GREEN + f"\n=== Alumnos con el nombre
'{nombre_busqueda}' ===")
            for alumno in alumnos:
                print(Fore.WHITE + f"ID: {alumno[0]}, Nombre:
{alumno[1]} {alumno[2]}, Edad: {alumno[3]}, Curso: {alumno[4]}, Email:
{alumno[5]}")

        else:
            print(Fore.RED + "[ERROR] Opción no válida.")
```

```
    except sqlite3.Error as e:
        print(Fore.RED + f"[ERROR] Ocurrió un problema al consultar la
base de datos: {e}")

    finally:
        conexion.close()
```

Esta función permite consultar alumnos en la base de datos mediante dos opciones: listar todos los alumnos registrados o buscar uno en particular por su nombre.

Al iniciar, se establece la conexión con la base de datos **instituto.db** y se muestra un menú con las dos opciones. Si el usuario elige la primera opción, se ejecuta una consulta SQL **SELECT * FROM alumnos**, recuperando todos los registros de la tabla y mostrándolos en la terminal con un formato claro.

Si se opta por la segunda opción, se solicita que se ingrese un nombre para buscar. Antes de realizar la consulta, el programa verifica que el nombre no esté vacío. Luego, se ejecuta una consulta parametrizada **SELECT * FROM alumnos WHERE nombre = ?** para recuperar sólo aquellos registros que coincidan con el nombre ingresado.

En ambos casos, si la base de datos no contiene registros o la búsqueda no arroja resultados, el programa muestra un mensaje informativo en color amarillo. Además, si ocurre un error en la consulta, se captura la excepción y se muestra un mensaje de error en color rojo. Para terminar, la conexión con la base de datos se cierra dentro del bloque **finally**, asegurando que no quede abierta después de la ejecución.

La función **eliminar_alumno()** implementa una confirmación antes de proceder con la eliminación. Se ha diseñado para que el usuario primero busque al alumno por su **ID** y luego confirme la eliminación antes de ejecutarla. Recordá que esto es sólo un ejemplo, y que vos podés, en tu TFI, implementar otras funcionalidades, o estas mismas pero de otra manera.

```
def eliminar_alumno():
    """
    Permite eliminar un alumno de la base de datos, asegurando que el
usuario confirme la acción antes de proceder.
```

```
    """

    # Conectar a la base de datos
    conexion = sqlite3.connect("instituto.db")
    cursor = conexion.cursor()

    try:
        print(Fore.CYAN + "\n=== Eliminar Alumno ===")

        # Mostrar todos los alumnos disponibles para que el usuario
pueda elegir
        cursor.execute("SELECT id, nombre, apellido, curso FROM
alumnos")
        alumnos = cursor.fetchall()

        if not alumnos:
            print(Fore.YELLOW + "[INFO] No hay alumnos registrados en la
base de datos.")
            return

        print(Fore.GREEN + "\n=== Lista de Alumnos ===")
        for alumno in alumnos:
            print(Fore.WHITE + f"ID: {alumno[0]}, Nombre: {alumno[1]}
{alumno[2]}, Curso: {alumno[3]}")

        # Solicitar el ID del alumno a eliminar
        id_alumno = input("\nIngresá el ID del alumno que querés
eliminar: ").strip()

        if not id_alumno.isdigit():
            print(Fore.RED + "[ERROR] El ID ingresado no es válido.")
            return

        id_alumno = int(id_alumno)

        # Verificar si el alumno existe en la base de datos
```

```
        cursor.execute("SELECT nombre, apellido FROM alumnos WHERE id =
?", (id_alumno,))
        alumno = cursor.fetchone()

        if not alumno:
            print(Fore.YELLOW + "[INFO] No se encontró un alumno con ese
ID.")
            return

        # Confirmación antes de eliminar
        confirmacion = input(Fore.YELLOW + f"¿Estás seguro de que querés
eliminar a {alumno[0]} {alumno[1]}? (S/N): ").strip().lower()

        if confirmacion != "s":
            print(Fore.GREEN + "[CANCELADO] La eliminación ha sido
cancelada.")
            return

        # Eliminar al alumno
        cursor.execute("DELETE FROM alumnos WHERE id = ?", (id_alumno,))
        conexion.commit()
        print(Fore.GREEN + f"[ÉXITO] Alumno {alumno[0]} {alumno[1]}
eliminado correctamente.")

    except sqlite3.Error as e:
        print(Fore.RED + f"[ERROR] Ocurrió un problema al eliminar el
alumno: {e}")

    finally:
        conexion.close()
```

La función comienza estableciendo una conexión con la base de datos **instituto.db**. Luego, recupera y muestra todos los alumnos registrados para que el usuario pueda identificar cuál desea eliminar. Se muestran los **ID**, **nombres** y **cursos** de los alumnos disponibles.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

A continuación, el usuario debe ingresar el **ID** del alumno a eliminar. Se verifica que el ID sea un número válido y que exista en la base de datos. Si el alumno no es encontrado, se muestra un mensaje informativo.

Si el alumno existe, el sistema solicita **una confirmación** antes de proceder con la eliminación. Se le pide al usuario que ingrese "S" para confirmar o cualquier otra tecla para cancelar la operación. Si se confirma, se ejecuta la sentencia **DELETE FROM alumnos WHERE id = ?**, eliminando el registro de la base de datos. Luego, el sistema confirma que la operación fue exitosa.

Si ocurre un error durante el proceso, como un problema en la base de datos, se captura y se muestra un mensaje de error en rojo. La conexión con la base de datos se cierra correctamente al finalizar la función.

Menú principal

Ahora, vamos a escribir el código del menú principal. Este menú permite elegir entre agregar, consultar y eliminar alumnos, asegurando una interacción intuitiva y clara con las funciones que escribimos antes.

```
def menu():
    """
    Muestra el menú principal y permite seleccionar las acciones a
realizar en la base de datos.
    """

    while True:
        print(Fore.BLUE + "\n=== Menú de Gestión de Alumnos ===")
        print(Fore.WHITE + "1. Registrar nuevo alumno")
        print(Fore.WHITE + "2. Consultar alumnos")
        print(Fore.WHITE + "3. Eliminar un alumno")
        print(Fore.WHITE + "4. Salir")

        opcion = input(Fore.CYAN + "Seleccioná una opción: ").strip()

        if opcion == "1":
```

```
            agregar_alumno()
        elif opcion == "2":
            consultar_alumnos()
        elif opcion == "3":
            eliminar_alumno()
        elif opcion == "4":
            print(Fore.GREEN + "\nGracias por usar el sistema. ¡Hasta
luego!")
            break
        else:
            print(Fore.RED + "[ERROR] Opción no válida. Intentá
nuevamente.")

# Ejecutar el menú
menu()
```

Este es la última pieza de nuestro proyecto. El **menú principal** es el punto de entrada del programa y permite al usuario navegar entre las diferentes opciones disponibles. Desde esta interfaz, podés registrar nuevos alumnos, consultar la base de datos, eliminar registros o salir del programa.

Podes probarlo poniendo en un mismo archivo **.py** todo el código que hemos visto, cuidando que las funciones estén al principio, y que se importen los módulos antes de usarlas. La última función a incluir es **menu()**.

Este programa permite registrar, consultar y eliminar alumnos, asegurando que la información ingresada sea válida y que las acciones realizadas sean claras para el usuario.

El uso de **try-except** garantiza que los errores en la base de datos sean manejados adecuadamente, evitando fallos inesperados y proporcionando mensajes informativos en caso de problemas. Colorama se emplea para mejorar la visualización de los mensajes en la terminal, destacando errores en rojo, información en amarillo y confirmaciones en verde, lo que facilita la comprensión de las operaciones realizadas.

Buenas prácticas y estructura del código

Cuando desarrollás un proyecto en Python, especialmente uno que involucra bases de datos y múltiples funciones, es fundamental que estructures el código de manera clara y ordenada. Un código bien organizado no sólo facilita su comprensión y mantenimiento, sino que también te permite realizar mejoras y agregar nuevas funcionalidades sin dificultad.

Una de las mejores prácticas es dividir el código en archivos según su función. En lugar de tener todo en un solo archivo, puedes separar la lógica del programa en módulos específicos. Por ejemplo, un archivo puede contener las funciones relacionadas con la base de datos, otro las funciones de interacción con el usuario y un tercero el menú principal. Esto ayuda a mantener cada parte del sistema bien delimitada, facilitando la depuración y el trabajo en equipo.
Otro aspecto clave es la legibilidad del código. Elegir nombres de variables y funciones descriptivos permite que tu código sea comprensible incluso para los demás. Usar convenciones como **snake_case** para los nombres de funciones y variables, así como mantener una correcta indentación, hace que el código sea más intuitivo. También es recomendable que evites redundancias y reutilices funciones en lugar de repetir líneas de código similares en diferentes partes del programa.

El manejo adecuado de errores es otro punto importante. Utilizar **try-except** para gestionar posibles fallos, como errores en la conexión a la base de datos o entradas incorrectas del usuario, mejora la estabilidad del sistema. Incluir mensajes de error claros y útiles evita que el usuario se frustre y proporciona información útil para corregir problemas.

Además, la implementación de comentarios en el código contribuye a su comprensión y mantenimiento. Agregar notas breves explicando qué hace cada función o bloque de código ayuda a quienes trabajan con el programa, incluyendo al propio desarrollador si necesita modificarlo en el futuro. Escribir documentación clara y concisa garantiza que el proyecto pueda ser utilizado y ampliado sin dificultades.

Si se siguen estas prácticas, el desarrollo de aplicaciones en Python será mucho más eficiente, permitiendo que los programas sean fáciles de entender, mantener y mejorar a lo largo del tiempo.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Presentación del proyecto

El **Trabajo Final Integrador** representa la culminación de todo lo aprendido en el curso. Más allá de la implementación técnica, es fundamental que el proyecto esté bien estructurado, correctamente documentado y presentado de una manera profesional. Esto no sólo facilita su comprensión y evaluación, sino que también te prepara para el mundo laboral, donde la claridad y la organización son tan importantes como el código mismo.

Organización del código:

Un código bien estructurado es fácil de leer, mantener y modificar. Para lograr esto, es recomendable dividir el código en funciones que realicen tareas específicas, evitando redundancias y asegurando una lógica clara. Además, la utilización de nombres descriptivos en variables y funciones mejora la comprensión del programa.

En el caso del TFI, tu aplicación podría estar organizada en módulos que se encarguen de la conexión con la base de datos, la manipulación de los datos y la interfaz con el usuario. Esto facilitaría su mantenimiento y -eventualmente- permite reutilizar funciones en distintos puntos del programa.

Uso de comentarios y documentación:

Un código sin comentarios es como un libro sin índice: difícil de entender y navegar. Es importante incluir comentarios explicativos en las secciones importantes del código, indicando la finalidad de cada función y los pasos más relevantes dentro de ellas. Sin embargo, los comentarios deben ser concisos y aportar valor, evitando redundancias.

Además de los comentarios en el código, se debe incluir un archivo de documentación, comúnmente llamado **README.txt**, en el que se expliquen aspectos fundamentales del proyecto. Este archivo debe incluir:

- Breve descripción del propósito de la aplicación.
- Instrucciones para la instalación y ejecución.
- Explicación de las funcionalidades implementadas.
- Cualquier requisito adicional para su correcto funcionamiento.

Para complementar la documentación, es recomendable incluir ejemplos de uso con capturas de pantalla o fragmentos de salida en la terminal. Esto ayuda a quienes revisen el código a entender rápidamente su funcionalidad sin necesidad de ejecutarlo. Por ejemplo, se pueden mostrar ejemplos de cómo se registran nuevos productos, cómo se buscan elementos en la base de datos y cómo se genera el reporte de bajo stock.

Uso de buenas prácticas

Además de modularizar el código, documentarlo correctamente y proporcionar ejemplos, es importante seguir buenas prácticas generales, tales como:

- Manejo de excepciones con **try-except** para evitar fallos inesperados.
- Uso de **Colorama** para mejorar la experiencia del usuario en la terminal.
- Validaciones de datos antes de ingresarlos en la base de datos para evitar inconsistencias.
- Implementación de un menú interactivo claro, con mensajes informativos y opciones intuitivas.

Siguiendo estos principios, el proyecto no solo cumplirá con los requisitos técnicos, sino que también será más robusto, fácil de entender y con un nivel de calidad que refleje el esfuerzo y aprendizaje adquirido a lo largo del curso.

Ejercicio Práctico

El día en **TalentoLab** está lleno de actividad. Mariana te llama a su oficina para una reunión breve.

"*Ahora que ya dominás SQLite, quiero que hagas algo importante antes de comenzar con tu TFI.*
Tu misión es escribir el programa que cree el archivo de la base de datos y la tabla que contendrá los datos. Esto te permitirá tener la estructura lista para cuando empieces a desarrollar las funcionalidades del TFI. Asegurate de que los campos sean los adecuados y de establecer correctamente las claves y restricciones."

Con el objetivo claro en mente, te dirigís a tu computadora, listo para escribir el primer código que dará vida a tu TFI.

Materiales y Recursos Adicionales:

Artículos:
Digital Talent Agency: 10 buenas prácticas para programadores
PY4E: Bases de datos y SQL

Videos:
DayiTecnologia: Creando un CRUD con SQLite3 y Python
LuisDev: Manejo de errores en Python

Preguntas para Reflexionar:

1. ¿Por qué es importante validar los datos antes de insertarlos en la base de datos? ¿Qué consecuencias podría tener no hacerlo?
2. En un proyecto real, ¿qué ventajas te ofrece modularizar tu código con funciones en lugar de escribir toda la lógica en un solo bloque de código?
3. Pensando en tu TFI, ¿qué funcionalidades planeás implementar primero y por qué? ¿Cómo podrías organizar tu trabajo para desarrollar el sistema de manera eficiente?
4. ¿Cómo podrías mejorar la experiencia del usuario en la interacción con tu sistema? ¿Qué elementos visuales o mensajes podrían hacer que la aplicación sea más clara y fácil de usar?

Próximos Pasos:

Después de todo el camino recorrido, llegamos a la etapa final del curso. Ya tenés todas las herramientas necesarias para desarrollar tu **Trabajo Final Integrador**, aplicando todo lo aprendido sobre estructuras de datos, bases de datos, validaciones, manejo de errores y buenas prácticas de programación.

En la próxima clase, vas a tener la oportunidad de **presentar tu proyecto**, compartir tu experiencia de desarrollo y recibir **retroalimentación** sobre tu trabajo. Será un espacio para reflexionar sobre los desafíos que enfrentaste, las decisiones que tomaste en tu código y cómo podrías seguir mejorando tu sistema.

También haremos un **repaso de los objetivos del curso**, destacando los conceptos fundamentales que adquiriste y su aplicación en proyectos reales. Además, te brindaremos orientación sobre cómo seguir aprendiendo y qué recursos adicionales podés explorar para profundizar tus conocimientos en Python y desarrollo de software.

Esta última clase no sólo marcará el **cierre del curso**, sino que será una instancia clave para consolidar lo aprendido y proyectar tus próximos pasos como programador. ¡Nos vemos en la próxima clase para celebrar tu progreso y compartir experiencias con el resto del equipo! 🚀

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad