

EVALUACIÓN

**PROGRAMACIÓN AVANZADA**  
Evaluación final

Michel Brevis

04-10-2024

Técnico en Análisis y  
Programación Computacional



**DESARROLLO:**

En una empresa de logística que gestiona el transporte de mercancías, se enfrentan a desafíos relacionados con la coordinación y seguimiento eficiente de los envíos. Actualmente, utilizan métodos manuales y sistemas dispersos que dificultan la gestión de la información y la comunicación entre los distintos equipos involucrados en el proceso logístico

Se requiere desarrollar una aplicación de escritorio utilizando Python y Tkinter para gestionar la información sobre los envíos de mercancías. La aplicación debe permitir a los usuarios realizar un seguimiento de los envíos, registrar información sobre las rutas y horarios de entrega, y generar informes sobre el estado de los envíos.

**Requerimientos:****Interfaz Gráfica:**

- La aplicación debe contar con una interfaz gráfica intuitiva que permita a los usuarios navegar fácilmente por las distintas funcionalidades.
- Debe incluir campos de entrada para ingresar información sobre los envíos, como número de seguimiento, origen, destino, fecha de entrega prevista, etc.
- Se debe proporcionar una lista o tabla que muestre los envíos registrados y su estado actual.

**Operaciones Básicas:**

- La aplicación debe permitir a los usuarios agregar nuevos envíos al sistema.
- Debe ser capaz de mostrar la lista de todos los envíos registrados, junto con su estado actual (en tránsito, entregado, etc.).
- Debe proporcionar la opción de actualizar la información de un envío existente, como el estado de entrega o la fecha estimada de entrega.

**Conexión a la Base de Datos:**

- La aplicación debe establecer una conexión a una base de datos MySQL local o remota para almacenar la información sobre los envíos.
- Se espera que utilice consultas SQL para realizar operaciones de lectura, escritura y actualización en la base de datos.

**Desarrollo del Código:**

- Se debe utilizar Python y Tkinter para desarrollar la interfaz gráfica de la aplicación.
- Deberán implementarse funciones para agregar, mostrar, actualizar y generar informes sobre los envíos.
- Se espera que se utilice el módulo `mysql.connector` para gestionar la conexión a la base de datos y realizar consultas SQL.

**Ejemplo de Creación de la Tabla e Inserción de Registros.**

```
CREATE TABLE Envios (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    NumeroSeguimiento VARCHAR(50) NOT NULL,  
    Origen VARCHAR(100) NOT NULL,  
    Destino VARCHAR(100) NOT NULL,  
    FechaEntregaPrevista DATE,  
    Estado VARCHAR(50) DEFAULT 'En tránsito'  
);  
INSERT INTO Envios (NumeroSeguimiento, Origen, Destino, FechaEntregaPrevista, Estado) VALUES  
    ('123456789', 'Ciudad A', 'Ciudad B', '2024-02-10', 'En tránsito'),  
    ('987654321', 'Ciudad B', 'Ciudad C', '2024-02-12', 'En tránsito'),  
    ('567890123', 'Ciudad C', 'Ciudad A', '2024-02-15', 'En tránsito');
```

**A continuación, responde las siguientes preguntas:**

**1. Explique a nivel teórico la aplicación práctica del control de flujo en Python. Además, ¿cuáles fueron las estructuras que utilizó en la solución de la empresa logística?**

El control de flujo en Python es fundamental porque nos permite tomar decisiones en nuestros programas y ejecutar diferentes bloques de código dependiendo de las condiciones que se presenten. Básicamente, lo que hacemos es decirle a la computadora "si pasa esto, ejecuta esto; si no pasa, ejecuta otra cosa".

A nivel práctico, en Python podemos controlar el flujo de ejecución usando estructuras como:

**Condicionales (if, elif, else):**

Nos permiten tomar decisiones en función de ciertas condiciones. Por ejemplo, si un número es mayor que otro, podemos hacer que el programa realice una acción específica.

**Para validar los datos ingresados por el usuario, como el número de seguimiento, el origen y destino del envío, y la fecha de entrega prevista. Por ejemplo, si algún campo está vacío, mostramos una advertencia al usuario.**

**Bucles (for, while):**

Nos permiten repetir un bloque de código varias veces. Un `for` se usa cuando sabemos cuántas veces queremos repetir algo, y un `while` se usa cuando queremos repetir hasta que se cumpla una condición.

**Se usó un bucle `for` para mostrar los envíos en la tabla. Cada vez que se agregan o actualizan envíos, recorremos la lista de envíos almacenados en la base de datos y los mostramos en la interfaz gráfica.**

**Excepciones (try, except):**

Son importantes para manejar errores en tiempo de ejecución. Si algo no sale como se esperaba, en lugar de que el programa falle, podemos manejar la situación.

**Se utilizó para manejar posibles errores que pueden ocurrir al intentar conectar con la base de datos o al ejecutar consultas SQL. Si algo falla, mostramos un mensaje de error en lugar de que el programa se detenga.**

En resumen, el control de flujo nos permite tomar decisiones, manejar errores y realizar acciones repetitivas dentro de nuestro programa, lo que es esencial para que funcione de manera lógica y eficiente.

## 2. Describa dos diferencias clave entre conjuntos y diccionarios. Explique si es o no adecuado utilizarlos en el caso de la aplicación de la empresa logística.

*En Python, tanto los conjuntos como los diccionarios son estructuras de datos muy útiles, pero tienen diferencias clave que determinan cuándo es adecuado utilizarlos.*

### Almacenamiento de datos:

- **Conjuntos:** Almacenan **valores únicos** sin ningún tipo de orden. Esto significa que no podemos tener elementos duplicados en un conjunto, y tampoco podemos acceder a un elemento usando un índice o clave, ya que no tiene un orden específico.
- **Diccionarios:** Almacenan pares **clave-valor**. Cada valor está asociado a una clave única, lo que nos permite acceder a los datos de manera directa utilizando la clave. Esto hace que los diccionarios sean ideales para cuando necesitamos una relación entre dos tipos de información.

### Acceso a los datos:

- **Conjuntos:** Solo permiten comprobar si un valor está presente o no. No hay un mecanismo para acceder directamente a un elemento específico. Son muy eficientes para operaciones como comprobar si algo ya existe en el conjunto.
- **Diccionarios:** Permiten un acceso directo a los valores a través de sus claves. Si tienes una clave, puedes obtener el valor asociado de manera rápida.

### Adecuación en la aplicación de la empresa logística:

- **Conjuntos:**

No serían muy adecuados para el caso de la empresa logística. La razón es que, en esta aplicación, necesitamos almacenar información detallada sobre cada envío (número de seguimiento, origen, destino, fecha de entrega, estado, etc.). Un conjunto solo nos permitiría tener una colección de datos únicos, pero no nos ofrece una manera clara de asociar la información de los envíos con sus atributos.

- **Diccionarios:**

Serían mucho más adecuados en ciertas partes de la aplicación. Por ejemplo, para almacenar la información de un envío, un diccionario sería útil porque podemos asociar cada atributo (como el número de seguimiento, origen y destino) con una clave. Esto nos permite acceder fácilmente a los datos y actualizarlos si es necesario.

3. Explique brevemente la funcionalidad de los módulos estándar como `collections`, `datetime`, `math` y `random`. Mencione un ejemplo de cómo utilizaría alguno de los módulos estudiados en el caso de la empresa.

En Python, los módulos estándar son bibliotecas integradas que proporcionan funcionalidades adicionales, evitando la necesidad de escribir código desde cero para tareas comunes. Algunos de los más útiles incluyen:

### Collections:

Este módulo ofrece tipos de datos especializados como `deque`, `Counter`, `defaultdict`, entre otros. Facilita la organización y el manejo de datos más complejos, como contar elementos de una lista o crear diccionarios con valores por defecto.

Un ejemplo, es que es útil para contar cuántas veces ocurre un elemento en una lista.

```
from collections import Counter
envios = ['transito', 'entregado', 'transito']
conteo_estados = Counter(envios)
print(conteo_estados) # Resultado: {'transito': 2, 'entregado': 1}
```

### Datetime:

Proporciona clases para trabajar con fechas y horas. Permite crear, manipular y formatear fechas, así como calcular diferencias entre ellas.

En la empresa logística, utilizaremos `datetime` para gestionar las fechas de entrega de los envíos. Por ejemplo, podemos calcular la diferencia entre la fecha actual y la fecha prevista de entrega para saber si un envío está retrasado.

```
from datetime import datetime
fecha_entrega = datetime(2024, 2, 10)
hoy = datetime.now()
diferencia = fecha_entrega - hoy
print(diferencia.days) # Calcula cuántos días faltan para la entrega
```

### Math:

Ofrece funciones matemáticas avanzadas como trigonometría, logaritmos, potencias, raíces, entre otras. Es útil para cálculos que requieren precisión o que involucran operaciones más complejas. Si necesitamos calcular distancias o realizar optimizaciones en rutas de entrega utilizando geometría, el módulo `math` nos facilita operaciones como la raíz cuadrada o el cálculo de ángulos.

```
import math
distancia = math.sqrt(16) # Resultado: 4.0
```

### Random:

Sirve para generar números aleatorios. Puede ser útil para simulaciones, pruebas, o para seleccionar elementos de manera aleatoria de una lista.

Podría usar `random` para seleccionar una muestra aleatoria de envíos a auditar, o para generar números de seguimiento únicos de prueba en un entorno de desarrollo.

```
import random
numero_seguimiento = random.randint(100000000, 999999999)
print(numero_seguimiento) # Resultado: un número aleatorio de 9 dígitos
```

### Ejemplo práctico en la empresa logística:

En el caso de la empresa logística, **utilizamos** `datetime` para gestionar las fechas de entrega de los envíos. Por ejemplo, al agregar un nuevo envío al sistema, el módulo `datetime` nos ayuda a manejar la fecha prevista de entrega y compararla con la fecha actual para ver si el envío está retrasado.

```
from datetime import datetime

fecha_entrega_prevista = datetime(2024, 2, 15)
fecha_actual = datetime.now()

if fecha_actual > fecha_entrega_prevista:
    print("El envío está retrasado.")
else:
    print("El envío está dentro del plazo.")
```

Este tipo de control es clave para monitorear el estado de los envíos y garantizar que la logística funcione de manera eficiente.

#### 4. ¿Cuáles son los principales elementos que deben examinarse al desarrollar e implementar una interfaz gráfica de usuario en el lenguaje de programación Python, tomando en consideración el caso de la empresa de logística?

- **Facilidad de Uso y Navegación Intuitiva**

La interfaz debe ser intuitiva, permitiendo que los usuarios interactúen fácilmente con las diferentes funcionalidades sin necesidad de capacitación avanzada. Los botones, menús y formularios deben estar organizados de manera clara y lógica.

En la aplicación de la empresa logística, los usuarios podrán agregar envíos, consultar el estado de un envío, y actualizar la información de manera fluida. Un mal diseño podría provocar errores o confusión al registrar envíos o generar informes.



- **Disposición y Organización de los Elementos en la Pantalla**

Los componentes como botones, campos de entrada, y tablas deben estar bien distribuidos en la ventana, evitando que se vea sobrecargada o desorganizada. La disposición debe ser coherente y seguir patrones visuales que faciliten su uso.

En el caso de la empresa de logística, los campos para ingresar información sobre los envíos (número de seguimiento, origen, destino, fecha de entrega) estarán organizados de manera que no confunda al usuario.

- **Validación de Datos**

Es crucial implementar validaciones para los datos que ingresan los usuarios. Esto previene errores y asegura que solo se ingresen datos válidos (como fechas en el formato correcto, números de seguimiento únicos, etc.)

La aplicación debe validar que los campos no queden vacíos o que se ingrese la fecha en un formato adecuado. Si la información no es válida, se mostrará un mensaje de advertencia para que el usuario corrija los errores antes de guardar los datos.

- **Feedback Visual y Mensajes de Error**

La aplicación debe proporcionar feedback al usuario, como mensajes de éxito al agregar un envío o advertencias cuando algo sale mal. Estos mensajes deben ser claros y proporcionar información suficiente para que el usuario sepa qué acción tomar.

Al agregar un nuevo envío, la aplicación mostrara un mensaje de confirmación. Si ocurre un error (como un problema de conexión con la base de datos), se mostrara un mensaje indicando qué falló y cómo resolverlo.

- **Conectividad con la Base de Datos**

La GUI debe estar conectada de manera eficiente con la base de datos para recuperar, almacenar y actualizar información en tiempo real. Esta conectividad debe ser estable y manejar errores de conexión de manera apropiada.

La aplicación de la empresa logística estará conectada con una base de datos MySQL para guardar los envíos y sus estados. Si la conexión falla, la interfaz lo manejarlo de manera adecuada, sin detener la ejecución completa de la aplicación.

- **Adaptabilidad y Escalabilidad**

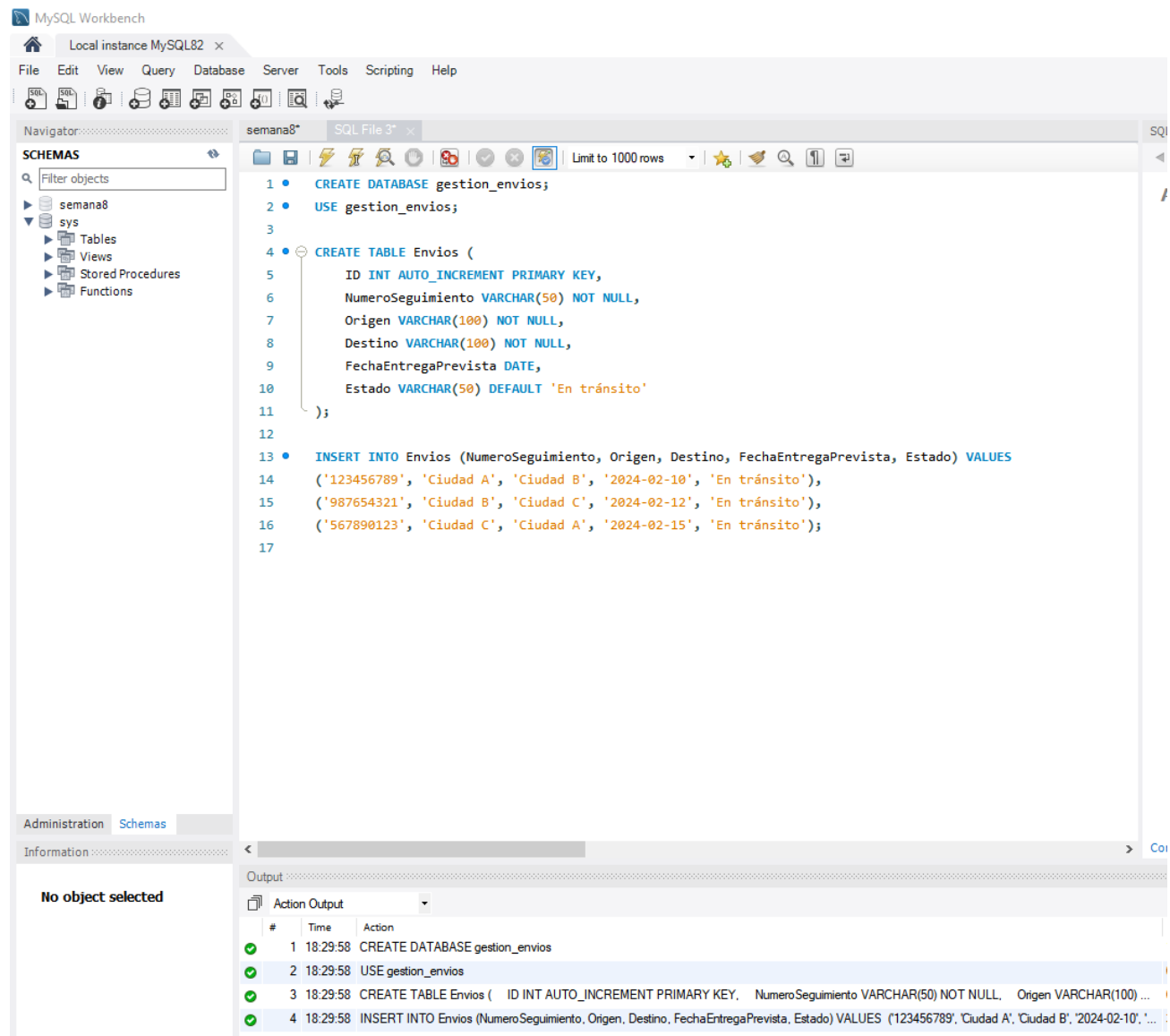
La interfaz debe ser adaptable para diferentes tamaños de pantalla y dispositivos, y escalable para futuras expansiones de la aplicación sin necesitar grandes cambios.

Si en el futuro se requieren nuevas funciones (como reportes más avanzados o seguimiento de envíos internacionales), la GUI se ajustará a estos cambios sin tener que rehacer todo el diseño desde cero.



5. Desarrolla una aplicación de escritorio en Python que cumpla con los requisitos mencionados en el problema. Asegúrate de proporcionar imágenes de evidencia que muestren tanto el código fuente como la interfaz gráfica creada, demostrando el funcionamiento de al menos una operación CRUD básica.

Para el desarrollo de la aplicación, primero creare la base de datos en SQL.



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'semana8' selected. The main editor window shows the following SQL code:

```
1 • CREATE DATABASE gestion_envios;
2 • USE gestion_envios;
3
4 • CREATE TABLE Envios (
5     ID INT AUTO_INCREMENT PRIMARY KEY,
6     NumeroSeguimiento VARCHAR(50) NOT NULL,
7     Origen VARCHAR(100) NOT NULL,
8     Destino VARCHAR(100) NOT NULL,
9     FechaEntregaPrevista DATE,
10    Estado VARCHAR(50) DEFAULT 'En tránsito'
11 );
12
13 • INSERT INTO Envios (NumeroSeguimiento, Origen, Destino, FechaEntregaPrevista, Estado) VALUES
14 ('123456789', 'Ciudad A', 'Ciudad B', '2024-02-10', 'En tránsito'),
15 ('987654321', 'Ciudad B', 'Ciudad C', '2024-02-12', 'En tránsito'),
16 ('567890123', 'Ciudad C', 'Ciudad A', '2024-02-15', 'En tránsito');
17
```

The bottom panel shows the 'Output' window with the 'Action Output' tab selected, displaying the execution results:

#	Time	Action
✓ 1	18:29:58	CREATE DATABASE gestion_envios
✓ 2	18:29:58	USE gestion_envios
✓ 3	18:29:58	CREATE TABLE Envios ( ID INT AUTO_INCREMENT PRIMARY KEY, NumeroSeguimiento VARCHAR(50) NOT NULL, Origen VARCHAR(100) ...
✓ 4	18:29:58	INSERT INTO Envios (NumeroSeguimiento, Origen, Destino, FechaEntregaPrevista, Estado) VALUES ('123456789', 'Ciudad A', 'Ciudad B', '2024-02-10', 'En tránsito', ...

Una vez verificada la creación de la tabla en SQL, procedo a conectar Python con MySQL.

```
import mysql.connector

def conectar():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="xxxxx",
        database="gestion_envios"
    )
```

Procedo con la creación de la interfaz gráfica con Tkinter, que permitirá agregar, mostrar y actualizar los envíos.

```
import tkinter as tk
from tkinter import messagebox, ttk
import mysql.connector
from mysql.connector import Error
from datetime import datetime

# Función para conectar a la base de datos MySQL
def conectar_db():
    try:
        conexion = mysql.connector.connect(
            host='localhost',
            database='EmpresaLogistica',
            user='root',
            password='SQL123*' # Coloca tu contraseña aquí
        )
        if conexion.is_connected():
            return conexion
    except Error as e:
        messagebox.showerror("Error de conexión", f"No se pudo conectar a la base de datos: {str(e)}")
        return None

# Función para agregar un envío a la base de datos
def agregar_envio():
    numero_seguimiento = entry_numero.get()
    origen = entry_origen.get()
    destino = entry_destino.get()
    fecha_entrega = entry_fecha.get()
    estado = combo_estado.get()

    if not numero_seguimiento or not origen or not destino or not fecha_entrega:
```

```
        messagebox.showwarning("Datos faltantes", "Por favor, completa todos los campos")
        return

    try:
        fecha_entrega_prevista = datetime.strptime(fecha_entrega, '%Y-%m-%d').date()
    except ValueError:
        messagebox.showwarning("Formato de fecha incorrecto", "La fecha debe estar en el formato AAAA-MM-DD")
        return

    conexion = conectar_db()
    if conexion:
        cursor = conexion.cursor()
        try:
            consulta = """INSERT INTO Envios (NumeroSeguimiento, Origen, Destino, FechaEntregaPrevista, Estado)
                            VALUES (%s, %s, %s, %s, %s)"""
            cursor.execute(consulta, (numero_seguimiento, origen, destino, fecha_entrega_prevista, estado))
            conexion.commit()
            messagebox.showinfo("Éxito", "El envío ha sido agregado exitosamente.")
            mostrar_envios()
        except Error as e:
            messagebox.showerror("Error", f"No se pudo agregar el envío: {str(e)}")
        finally:
            cursor.close()
            conexion.close()

# Función para mostrar los envíos en la tabla
def mostrar_envios():
    for fila in treeview.get_children():
        treeview.delete(fila)

    conexion = conectar_db()
    if conexion:
        cursor = conexion.cursor()
        try:
            cursor.execute("SELECT * FROM Envios")
            resultados = cursor.fetchall()
            for envio in resultados:
                treeview.insert('', tk.END, values=envio)
        except Error as e:
            messagebox.showerror("Error", f"No se pudieron obtener los envíos: {str(e)}")
        finally:
```

```
        cursor.close()
        conexion.close()

# Función para actualizar el estado de un envío
def actualizar_envio():
    try:
        selected_item = treeview.selection()[0]
        envio_id = treeview.item(selected_item)['values'][0]
        nuevo_estado = combo_actualizar_estado.get()

        conexion = conectar_db()
        if conexion:
            cursor = conexion.cursor()
            consulta = "UPDATE Envios SET Estado = %s WHERE ID = %s"
            cursor.execute(consulta, (nuevo_estado, envio_id))
            conexion.commit()
            messagebox.showinfo("Éxito", "El estado del envío ha sido
actualizado.")
            mostrar_envios()
        except IndexError:
            messagebox.showwarning("Selección requerida", "Por favor, selecciona un
envío de la lista para actualizar.")
        except Error as e:
            messagebox.showerror("Error", f"No se pudo actualizar el envío: {str(e)}")

# Crear la ventana principal
ventana = tk.Tk()
ventana.title("Sistema de Gestión de Envíos")
ventana.geometry("800x600")

# Etiquetas y campos de entrada para agregar envíos
frame_formulario = tk.Frame(ventana)
frame_formulario.pack(pady=20)

tk.Label(frame_formulario, text="Número de Seguimiento").grid(row=0, column=0,
padx=10, pady=5)
entry_numero = tk.Entry(frame_formulario)
entry_numero.grid(row=0, column=1, padx=10, pady=5)

tk.Label(frame_formulario, text="Origen").grid(row=1, column=0, padx=10, pady=5)
entry_origen = tk.Entry(frame_formulario)
entry_origen.grid(row=1, column=1, padx=10, pady=5)

tk.Label(frame_formulario, text="Destino").grid(row=2, column=0, padx=10, pady=5)
entry_destino = tk.Entry(frame_formulario)
entry_destino.grid(row=2, column=1, padx=10, pady=5)
```

```
tk.Label(frame_formulario, text="Fecha de Entrega (AAAA-MM-DD)").grid(row=3,
column=0, padx=10, pady=5)
entry_fecha = tk.Entry(frame_formulario)
entry_fecha.grid(row=3, column=1, padx=10, pady=5)

tk.Label(frame_formulario, text="Estado").grid(row=4, column=0, padx=10, pady=5)
combo_estado = ttk.Combobox(frame_formulario, values=["En tránsito", "Entregado"])
combo_estado.grid(row=4, column=1, padx=10, pady=5)
combo_estado.current(0)

# Botón para agregar envío
tk.Button(frame_formulario, text="Agregar Envío",
command=agregar_envio).grid(row=5, column=0, columnspan=2, padx=10, pady=10)

# Tabla para mostrar los envíos
frame_tabla = tk.Frame(ventana)
frame_tabla.pack(pady=20)

treeview = ttk.Treeview(frame_tabla, columns=("ID", "Número", "Origen", "Destino",
"Fecha", "Estado"), show='headings')
treeview.heading("ID", text="ID")
treeview.heading("Número", text="Número de Seguimiento")
treeview.heading("Origen", text="Origen")
treeview.heading("Destino", text="Destino")
treeview.heading("Fecha", text="Fecha de Entrega")
treeview.heading("Estado", text="Estado")
treeview.pack()

# Botón para actualizar estado
frame_actualizar = tk.Frame(ventana)
frame_actualizar.pack(pady=20)

tk.Label(frame_actualizar, text="Actualizar Estado").grid(row=0, column=0, padx=10,
pady=5)
combo_actualizar_estado = ttk.Combobox(frame_actualizar, values=["En tránsito",
"Entregado"])
combo_actualizar_estado.grid(row=0, column=1, padx=10, pady=5)
combo_actualizar_estado.current(0)

tk.Button(frame_actualizar, text="Actualizar Envío",
command=actualizar_envio).grid(row=1, column=0, columnspan=2, padx=10, pady=10)

# Mostrar envíos al iniciar
mostrar_envios()

# Iniciar el loop principal
ventana.mainloop()
```

Al ejecutar la aplicación, se ingresará un nuevo envío en los campos de texto que aparecen en la parte superior y hacer clic en el botón "Agregar Envío". Este envío será guardado en la base de datos y aparecerá en la tabla.

Sistema de Gestión de Envíos

Número de Seguimiento: 123

Origen: Maipu

Destino: santiago

Fecha de Entrega (AAAA-MM-DD): 2024-10-03

Estado: En tránsito

Agregar Envío

ID	Número de Seguimiento	Origen	Destino
1	123456789	Ciudad A	Ciudad B
2	987654321	Ciudad B	Ciudad C
3	567890123	Ciudad C	Ciudad A
4	123	Maipu	santiago

Actualizar Estado: Entregado

Actualizar Envío

**REFERENCIAS BIBLIOGRÁFICAS:**

- **Ejemplo texto de lectura de IACC:**

IACC. (2024). *Programación avanzada*

Todos los contenidos, de la semana 1 a 8