# Informe Final - Gestor de Turnos

## 1. Portada

**Proyecto:** Gestor de Turnos  
**Materia:** [Nombre de la materia]  
**Carrera:** [Nombre de la carrera]  
**Año:** 2025  
**Integrantes:**  
- Juan Domínguez

**Tema:** Desarrollo de una aplicación de escritorio para la gestión de turnos utilizando Python, interfaz moderna y principios de programación orientada a objetos (POO).

**Motivación:** La organización de turnos es una necesidad común en rubros como peluquerías, consultorios o talleres. Este proyecto nace del objetivo de desarrollar una solución funcional, moderna y escalable que permita automatizar esta tarea de forma simple, integrando conocimientos adquiridos durante la cursada.

## 2. Metodología y Desarrollo

El proyecto se abordó siguiendo una metodología incremental y modular. Primero se definieron los elementos clave: cliente, servicio y turno. A partir de ahí se fueron construyendo clases y componentes independientes que luego se integraron entre sí.

### Decisiones técnicas destacadas:

* Se utilizó **Python 3.11** como lenguaje base.
* Se aplicó **Programación Orientada a Objetos (POO)** para representar las entidades del dominio.
* Se empleó el patrón de diseño **Strategy** para el sistema de notificaciones.
* Se eligió la biblioteca **customtkinter** para la interfaz gráfica por su diseño moderno y accesibilidad.
* Se optó por **JSON** como formato de almacenamiento persistente.
* Para exportar datos, se utilizó la librería **openpyxl**, que permite generar archivos Excel.

### Etapas del desarrollo:

1. Modelado de clases base (Cliente, Servicio, Turno).
2. Lógica de gestión con validaciones (GestorTurnos).
3. Implementación de persistencia en JSON.
4. Construcción de la interfaz visual con customtkinter.
5. Incorporación de helpers y mejoras visuales.

## 3. Diseño y Estructura del Código

El proyecto se organizó en carpetas según responsabilidades:

* /modelos: clases que representan entidades del sistema.
* /gestor: clase principal de control (GestorTurnos).
* /interfaz: componentes visuales.
* /notificaciones: implementaciones del patrón Strategy.
* /utils: funciones auxiliares (ID, persistencia, helpers).
* /datos: archivo persistente turnos.json.

### Clases principales:

* Cliente: almacena nombre, contacto, ID.
* Servicio: almacena nombre y duración.
* Turno: vincula cliente, servicio, fecha y hora.
* GestorTurnos: contiene la lógica principal de asignación, validación, eliminación y exportación.
* Notificador: estrategia de notificación (simulada).

### Patrón aplicado:

* **Strategy**: para permitir notificaciones distintas sin cambiar la lógica de turnos.

## 4. Resultados y Dificultades

### Resultados obtenidos:

* Se logró una aplicación funcional, estable y modular.
* Interfaz clara, intuitiva y estéticamente moderna.
* Exportación a Excel con datos completos.
* Datos persistentes entre sesiones.

### Dificultades enfrentadas:

* Manejo de errores por archivo JSON vacío: se resolvió con validación en persistencia.py.
* Compatibilidad entre formatos de datos (str vs. objetos): se solucionó con reconstrucción de objetos desde diccionarios.
* Integración de la GUI: se decidió migrar a customtkinter para mayor calidad visual.

## 5. Conclusiones y Reflexiones

Este proyecto permitió aplicar de forma concreta y funcional conceptos fundamentales de la programación orientada a objetos, el diseño modular y el uso de interfaces gráficas. Además, se practicó la lectura y escritura de datos persistentes, el uso de librerías externas y la escritura de código mantenible.

Fue una experiencia completa que integró varias de las herramientas vistas durante la cursada. Se evidenció la importancia de planificar la estructura desde el inicio y de probar cada componente de forma aislada.

### Futuras mejoras posibles:

* Agregar selector de calendario.
* Permitir turnos recurrentes.
* Integrar una base de datos como SQLite.
* Agregar soporte para notificaciones reales por email o WhatsApp.

**Fin del informe**