Informe de Proyecto: Sistema de Registro de Ventas y Generación de Boletas

Integrantes del Grupo: [Nombres de los Integrantes Aquí]

17 de noviembre de 2025

1 Descripción del Problema o Necesidad Detectada

La necesidad principal detectada es la **falta de un sistema automatizado y estandarizado para registrar transacciones de venta** en un entorno pequeño o microempresa (como una tienda o puesto de mercado).

Actualmente, el proceso se realiza de forma manual, lo que conlleva varios problemas:

- **Errores de Cálculo:** Susceptibilidad a errores humanos al sumar subtotales y calcular el total de la venta.
- **Falta de Trazabilidad:** Dificultad para rastrear ventas pasadas, lo que complica la contabilidad o la gestión de devoluciones.
- **Recibos Informales:** Los recibos (boletas) generados no son uniformes ni incluyen detalles importantes como fecha y hora.

2 Explicación de la Solución Propuesta

La solución propuesta es un **Sistema de Registro de Ventas basado en Python** que opera a través de la terminal, pero que integra una interfaz gráfica ligera para la visualización inmediata del comprobante.

2.1 ¿Qué hace el programa?

- **Ingreso Interactivo:** Permite al usuario (vendedor) ingresar el nombre del cliente y los productos vendidos en un formato simple (‘cantidad:producto‘).
- **Cálculo Automático:** Calcula el subtotal de cada ítem y el total general de la venta.
- **Validación de Datos:** Verifica que la cantidad sea un número positivo y que el producto exista en el inventario.
- **Generación de Boleta:** Una vez finalizada la venta, genera dos salidas:
    1. Un archivo de texto (‘.txt‘) con la boleta completa, guardado en un directorio ‘boletas/‘.
    2. Una ventana gráfica (`tkinter`) que muestra la boleta inmediatamente para que el vendedor pueda consultarla o imprimirla si es necesario.

2.2 Audiencia y Justificación de la Idea

El programa está dirigido a **pequeños comerciantes, vendedores ambulantes o estudiantes** que necesitan una herramienta sencilla y de bajo costo para formalizar sus procesos de venta. Se eligió esta idea para demostrar la integración de conceptos clave del curso: manejo de archivos, módulos de Python y una introducción a la interfaz gráfica del usuario con `tkinter`.

3 Capturas de Pantalla del Código y Conceptos Aplicados

A continuación, se presentan extractos clave del código con una indicación clara de los conceptos vistos en clase aplicados.

3.1 Estructuras de Control y Modularidad (Funciones)

El uso de funciones permite la reutilización de código y la organización modular, como se ve en la generación de la boleta:

```
def generar_contenido_boleta(cliente, productos_cliente, total, fecha):
    """Genera el contenido de la boleta para reutilizar en archivo y GUI."""
    lineas = [
        f"Boleta para {cliente}", # Formato de string f-string
        f"Fecha y Hora: {fecha}",
        "\nDetalles de la Venta:\n"
    ]

    for cantidad, producto in productos_cliente: # Bucle For para iteración
        precio = PRODUCTOS_DISPONIBLES[producto]
        subtotal = cantidad * precio
        lineas.append(f"{cantidad} x {producto} a ${precio:.2f} c/u => Total: ${subtotal:.2f

    lineas.append(f"\nTOTAL: ${total:.2f}")
    return "\n".join(lineas)
```

3.2 Manejo de Errores y Excepciones

La función de validación de entrada utiliza bloques `try/except` para asegurar que el programa no se detenga por errores del usuario (ej. ingresar texto en lugar de números o formatos incorrectos). Este es un concepto fundamental de robustez.

```
def validar_entrada_producto(entrada):
    """Valida y procesa la entrada del producto."""
    try:
        partes = entrada.split(":")
        if len(partes) != 2:
            raise ValueError("Formato incorrecto")

        cantidad_str, producto = partes
        cantidad = int(cantidad_str.strip()) # Conversión de tipo (casting)

        if cantidad <= 0:
            raise ValueError("La cantidad debe ser mayor a 0")

        if producto not in PRODUCTOS_DISPONIBLES:
            raise ValueError(f"Producto '{producto}' no encontrado")

        return cantidad, producto

    except ValueError as e: # Captura de excepciones (Concepto de try/except)
        return None, str(e)
```

3.3 Manejo de Archivos

El registro de la boleta utiliza el manejo seguro de archivos y el módulo `pathlib` para crear directorios y rutas de archivos de manera independiente del sistema operativo.

```
# Inicialización y creación de directorio:
BOLETAS_DIR = Path("boletas") # Uso de pathlib
BOLETAS_DIR.mkdir(exist_ok=True) # Creación segura de directorio

def registrar_boleta(cliente, productos_cliente, total):
    # ... código omitido ...

    ruta_boleta = BOLETAS_DIR / boleta_nombre
    try:
        # Uso de "with open(...)" para asegurar el cierre del archivo
        with open(ruta_boleta, "w", encoding="utf-8") as boleta:
            boleta.write(contenido)
    except IOError as e:
        # Manejo de excepción de Entrada/Salida
        messagebox.showerror("Error", f"No se pudo guardar la boleta: {e}")
        return
    # ... código omitido ...
```

4 Conclusiones del Grupo

4.1 Aprendizajes Obtenidos

- **Integración de Módulos:** Logramos combinar la lógica de la consola con la funcionalidad de la GUI (`tkinter`) y el manejo de archivos (`pathlib`).
- **Robustez del Código:** Comprendimos la importancia de la validación de entrada y la gestión de excepciones (`try/except`) para evitar fallos inesperados.
- **Manejo de Fechas:** Se aprendió a utilizar el módulo `datetime` para registrar la fecha y hora de la transacción, un requisito clave en cualquier sistema comercial.

4.2 Dificultades Encontradas

- **Interacción de `tkinter` y Consola:** Coordinar el flujo de la terminal (`realizar_venta`) con el bucle principal de la ventana gráfica (`mainloop`) requirió cuidado para que la aplicación no se bloqueara. Tuvimos que asegurar que la ventana de la boleta funcionara como un elemento temporal y no interrumpiera el flujo principal de la consola.
- **Formato de Archivos:** Diseñar una estructura de nombre de archivo segura (función `obtener_nombre_boleta`) que funcione en diferentes sistemas operativos (Windows, Linux, macOS) fue un pequeño desafío.

4.3 Posibles Mejoras

- **Inventario Dinámico:** Conectar el sistema a una base de datos (como `SQLite` o `Firestore`) para gestionar el inventario de manera persistente en lugar de un diccionario estático.
- **Interfaz Gráfica Principal:** Implementar la lógica de venta completamente en `tkinter` para ofrecer una experiencia más profesional y fácil de usar.
- **Reportes:** Añadir una función para leer las boletas guardadas y generar reportes de ventas diarios o mensuales.

5 Justificación de Incorporación de Elementos o Librerías no Vistas en Clase

El proyecto incorpora tres módulos estándar de Python que pudieron no haber sido vistos en detalle en clase, pero que son esenciales para un desarrollo moderno y robusto:

- `tkinter` **(Módulo Estándar):** Se utilizó para la función `mostrar_boleta_ventana`. La justificación es que el requisito era generar un comprobante, y una ventana gráfica permite al vendedor **ver y potencialmente imprimir** el recibo de forma visualmente clara, lo cual no es posible solo con la consola.
- `pathlib` **(Módulo Estándar):** Se utilizó en lugar del módulo `os` para el manejo de directorios y rutas. `pathlib` ofrece una sintaxis más intuitiva y orientada a objetos para construir rutas de archivo (`BOLETAS_DIR / boleta_nombre`), lo que mejora la legibilidad del código y su compatibilidad cruzada de sistemas operativos.
- `f-strings` **(Sintaxis de Python):** Se utilizó ampliamente para el formateo de cadenas de texto y la creación de la boleta. Esta sintaxis es el método preferido y más eficiente en Python moderno para incluir variables dentro de cadenas, mejorando la claridad sobre el método `.format()`.

6 Link al Repositorio del Proyecto

El repositorio del proyecto, donde todos los integrantes tienen commits registrados, se encuentra en el siguiente enlace:

[INSERTAR AQUÍ EL LINK AL REPOSITORIO (GitHub/GitLab)]