Profesor: Soledad Demicheri

Alumnos:

Yair Bustamante

Gonzalo Bergmans

Juan Pablo Turco

Guillermo Damiani

Fecha de entrega: 18/11/2025

Contenidos

Informe de proyecto: Sistema de registro de ventas y Generación de boletas

1. Descripción del Problema

La necesidad detectada es la falta de un sistema automatizado y estandarizado para registrar transacciones de venta en un entorno pequeño o microempresa. Posibles errores que se presentan al momento de hacer la carga de datos manual:

- Errores de Cálculo: Evitar errores al sumar subtotales y calcular el total de la venta.
- Registros: Dificultad para tener un registro de ventas pasadas.
- Recibos informales: Los recibos generados no tienen detalles importantes, por ejemplo, fecha y hora.

2. Explicación de la solución propuesta

Solución propuesta: Sistema de registro de ventas basado en Python que opera a través de la interfaz gráfica.

3. ¿Qué hace el programa?

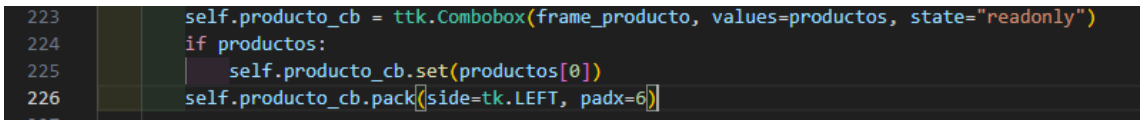
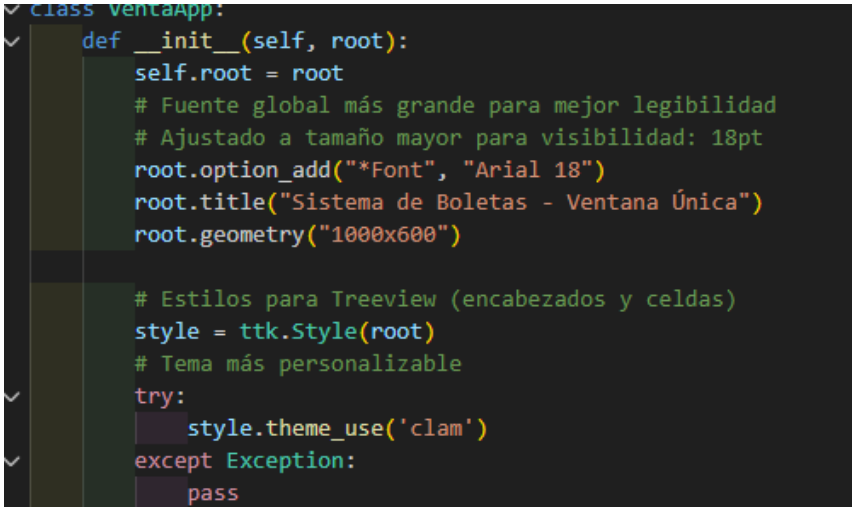
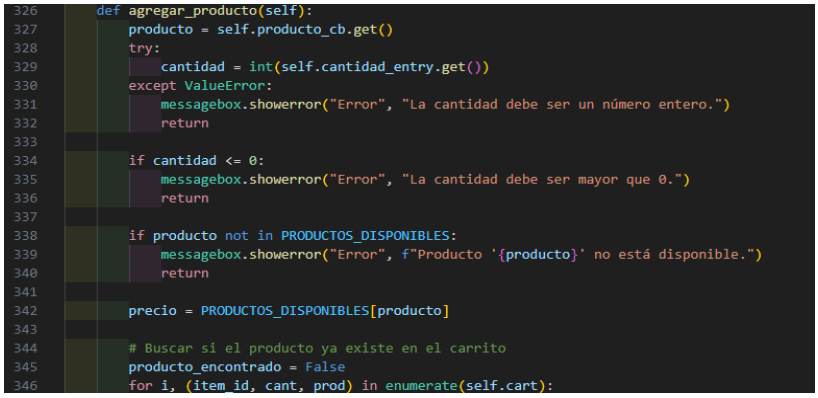
- Ingreso interactivo: Permite al usuario (vendedor) ingresar el nombre del cliente y los productos vendidos de forma intuitiva.
- Cálculo automático: Calcula el subtotal de cada ítem y el total general de la venta.
- Validación de datos: Verifica que la cantidad sea un número positivo y que el producto exista en el inventario.
- Generación de boleta: Al finalizar la venta, se genera la boleta con dos salidas:
    - Archivo de texto (.txt) con la boleta completa, guardada en una carpeta llamada “boletas/”.
    - La interfaz gráfica creada con tkinter en donde el vendedor va a poder cargar los datos del cliente.

4. Justificación de la idea

Está dirigido a pequeños comerciantes, vendedores ambulantes o estudiantes que necesitan una herramienta sencilla y de bajo costo para formalizar los procesos de venta.

5. Capturas de pantalla del código y conceptos aplicados

Unidad 1 y 2:

- Entrada:
  - Nombre del cliente:
  - 
  - Cantidad de productos (cargados en el archivo .txt)
  - Productos
  - 
  - Fecha y hora:
  - 
- Proceso:
  - Generar interfaz (líneas 166 – 310)
  - 
  - Pedir entradas
  - Calcular precio total de los productos (línea 326 – 368)
  - 
  - Eliminar un producto no deseado

```
480     def guardar_boleta(self):
481         cliente = self.cliente_entry.get().strip()
482         if not cliente:
483             messagebox.showerror("Error", "Ingrese el nombre del cliente.")
484             return
```

- Generar archivo resumen al terminar la boleta
- Generar y mostrar código QR, donde se guarda la información de la boleta.

- Salida:
  - Precio unitario de cada producto y el precio unitario por la cantidad según el archivo “lista_de_producto.txt”
  -
```
234         self.producto_cb.bind('<<ComboboxSelected>>', self.mostrar_precio_seleccionado)
```
  - Precio total de la compra
  -
```
productos_pa (variable) cantidad: Any oducto) for _id, cantidad, product in self.cart for producto in [product]]
total = sum(cantidad * PRODUCTOS_DISPONIBLES[producto] for cantidad, producto in productos_para_guardar)
try:
    boleta_nombre, contenido = registrar_boleta(cliente, productos_para_guardar, total)
except Exception as e:
    messagebox.showerror("Error", f"No se pudo guardar la boleta: {e}")
    return
```
  - Cliente_fecha_hora
  - Código QR
  -
```
# Intentar generar un código QR con el contenido de la boleta
try:
    # si la importación falló antes, intentar importar localmente (por si se instaló luego)
    qr_lib = qrcode
    if qr_lib is None:
        try:
            import qrcode as _qrcode
            qr_lib = _qrcode
        except Exception:
            qr_lib = None

    if qr_lib is not None:
        img = qr_lib.make(contenido)
        ruta_qr = ruta_boleta.with_suffix('.png')
        try:
            img.save(ruta_qr)
        except Exception as save_err:
            print(f"No se pudo guardar el QR: {save_err}")
except Exception as e:
    print(f"No se generó el QR: {e}")
```

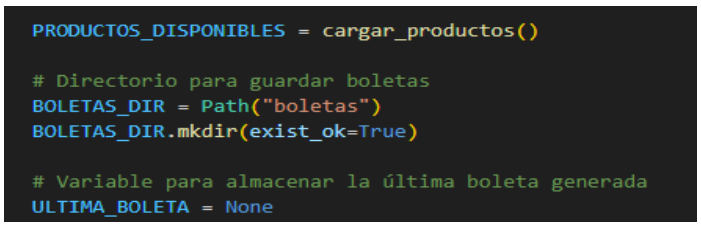
Unidad 3:

Se utilizó el control de versiones (GitHub) para desarrollar este programa en donde cada integrante subió las modificaciones propias.

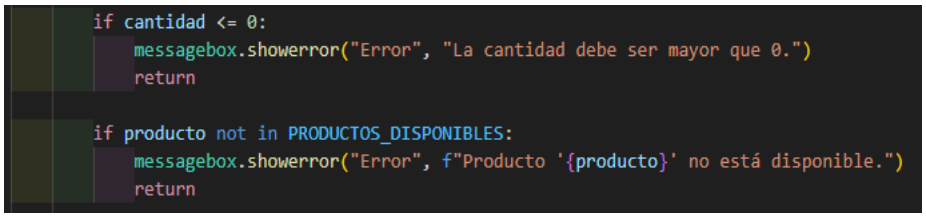
Unidad 4:

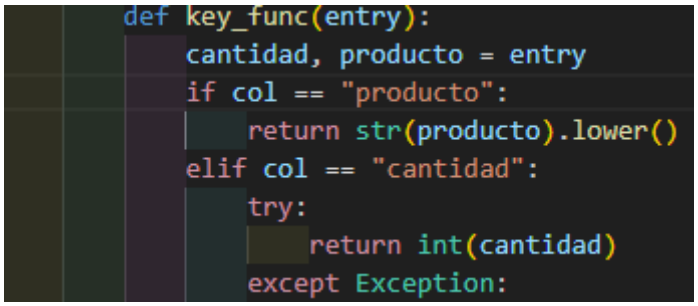
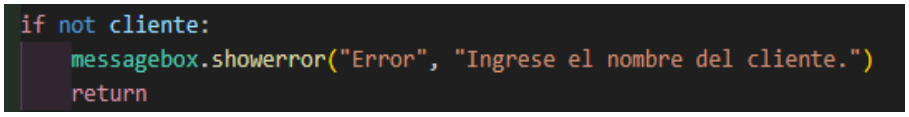
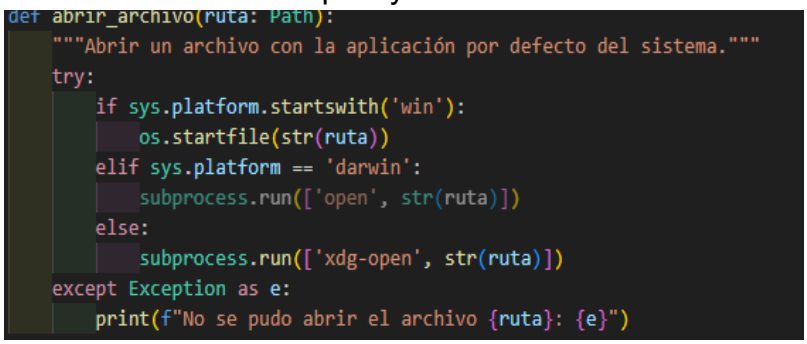
- Se utilizaron variables y el tipo de datos

Variable	Línea
PRODUCTOS_DISPONIBLES (dict)	49
self.cart (lista)	201
self.qr_photo (imagen)	290
cliente (str)	392
cantidad (int)	329

- Se utilizó una constante para guardar la ruta de acceso a las boletas generadas
  - 
```
PRODUCTOS_DISPONIBLES = cargar_productos()

# Directorio para guardar boletas
BOLETAS_DIR = Path("boletas")
BOLETAS_DIR.mkdir(exist_ok=True)

# Variable para almacenar la última boleta generada
ULTIMA_BOLETA = None
```
- Se utilizan expresiones lógicas, booleanas y relacionales a lo largo del código
  - 
```
if cantidad <= 0:
    messagebox.showerror("Error", "La cantidad debe ser mayor que 0.")
    return

if producto not in PRODUCTOS_DISPONIBLES:
    messagebox.showerror("Error", f"Producto '{producto}' no está disponible.")
    return
```
  - 
```
def key_func(entry):
    cantidad, producto = entry
    if col == "producto":
        return str(producto).lower()
    elif col == "cantidad":
        try:
            return int(cantidad)
        except Exception:
```
  - 
```
if not cliente:
    messagebox.showerror("Error", "Ingrese el nombre del cliente.")
    return
```
- Se utilizaron condiciones simples y dobles
  - 
```
def abrir_archivo(ruta: Path):
    """Abrir un archivo con la aplicación por defecto del sistema."""
    try:
        if sys.platform.startswith('win'):
            os.startfile(str(ruta))
        elif sys.platform == 'darwin':
            subprocess.run(['open', str(ruta)])
        else:
            subprocess.run(['xdg-open', str(ruta)])
    except Exception as e:
        print(f"No se pudo abrir el archivo {ruta}: {e}")
```
- Se utilizó el “debug” para comprobar y ver los errores que se fueron dando a lo largo del desarrolló del código.

Unidad 5

- Se utilizó en todo el código el ciclo for en varias oportunidades.

-
```
try:
    with open("lista_de_productos.txt", "r", encoding="utf-8") as f:
        for linea in f:
            linea = linea.strip()
            # Saltar líneas vacías y categorías (que comienzan y terminan con -)
            if not linea or (linea.startswith("-") and linea.endswith("-")):
                continue
```

- Uso de un acumulador para el total y un contador para la cantidad de productos
  - Acumulador:
    ```
    383                 total += cantidad * precio
    ```
  - Contador:
    ```
        total = 0
        for _id, cantidad, producto in self.cart:
            precio = PRODUCTOS_DISPONIBLES.get(producto, 0)
            total += cantidad * precio
    ```

Unidad 6

- Uso de vectores para cargar el contenido
  - ```
    self.cart = []  # lista de (cantidad, producto)
    ```
- Uso de diccionario para registrar clientes
  - ```
    PRODUCTOS_DISPONIBLES = cargar_productos()
    ```
- Modificación de cadenas
  - ```
    f"Boleta para {cliente}",
    f"Fecha y Hora: {fecha}",
    "\nDetalles de la Venta:\n"
    ```
- Matriz
  - ```
    productos_para_qr = [(cantidad, producto) for _id, cantidad, prod in self.cart for producto in [prod]]
    ```

Unidad 7

- Utilización de elementos aprendidos en esta unidad para segmentar el código en partes, logrando que se trabaje de manera fluida y limpiamente.
- Uso de funciones y un segmento para mostrar la lista de productos.

Unidad 8

- Uso del sistema de ordenamiento con Inserción para encontrar los productos con menor y mayor cantidad, precio unitario o subtotal.

-
```
# selection sort sobre la lista
n = len(entries)
for i in range(n):
    sel = i
    for j in range(i + 1, n):
        a = key_func(entries[j])
        b = key_func(entries[sel])
        if asc:
            if a < b:
                sel = j
        else:
            if a > b:
                sel = j
    if sel != i:
        entries[i], entries[sel] = entries[sel], entries[i]
```

Unidad 9

- Uso de archivos .txt para la lista de productos y las boletas. Además de un .png para guardar los códigos QR.
- Lectura y/o edición de archivos para buscar datos, guardar información.
  - Lectura del archivo .txt:
```
productos = {}
try:
    with open("lista_de_productos.txt", "r", encoding="utf-8") as f:
```
  - Escritura del archivo .txt:
```
try:
    with open(ruta_boleta, "w", encoding="utf-8") as boleta:
        boleta.write(contenido)
except IOError as e:
```

Conclusión

Aprendizajes nuevos:

- Integración de módulos: Pudimos desarrollar de una manera más que óptima una interfaz gráfica con tkinter y mejorar el manejo de archivos.
- Robustez del código: Buscamos de todas las maneras que se nos ocurrió para romper nuestro código e ir haciéndole mejoras con esos detalles.
- Manejo de fechas: Aprendimos a utilizar la librería datetime para obtener el la fecha y hora.
- QRCODE: Aprender del uso de esta librería que se implemento para que al final al usuario le deje escanear y muestre el contenido de la boleta.

Dificultades

- Varios errores al momento de ir ejecutando el código de una computadora a otra, debido que a veces surgía problema con las librerías por el hecho de que no estaban instaladas, y si pasaba eso, no se instalaban en la ruta donde está instalado Python.

- Aplicar el método de ordenamiento con una sola clase que tuvimos, que solamente fue teoría y de que se renegó bastante con la manera de estructurar el código en base a archivos .txt

Elementos incorporados no vistos

- Tkinter: Utilizada para generar la interfaz gráfica, se hizo lo mas “amigable” posible hacia la vista del vendedor sin quitarle practicidad o añadir datos innecesarios.
- Pathlib: Utilizada en lugar del módulo o para el manejo de directorios y rutas. Mejora la legibilidad del código.
- f-strings: No lo utilizamos mucho en clase, pero está implementado en el código para el formateo de cadenas de textos y la creación de la boleta. Método mas eficiente para incluir variable dentro de cadenas.
- qrcode: Con esta librería nos permitió generar el código QR que pertenece a cada boleta.

Link al repositorio del proyecto y al video

- Repositorio: [https://github.com/YairEB46/-TPI-PROGRAMACI-N-2025-.git]
- Video: [https://youtu.be/LOEP1dBGzJY]