Benemérita Universidad Autónoma de Puebla

Ingeniería en Ciencia de Datos

# Programación Avanzada

Carlos Alexander Castillo Gomez

Prof. Jaime Alejandro Romero Sierra

## Actividad: cafetería

Primavera 2025

Fecha de entrega:21/04/2025

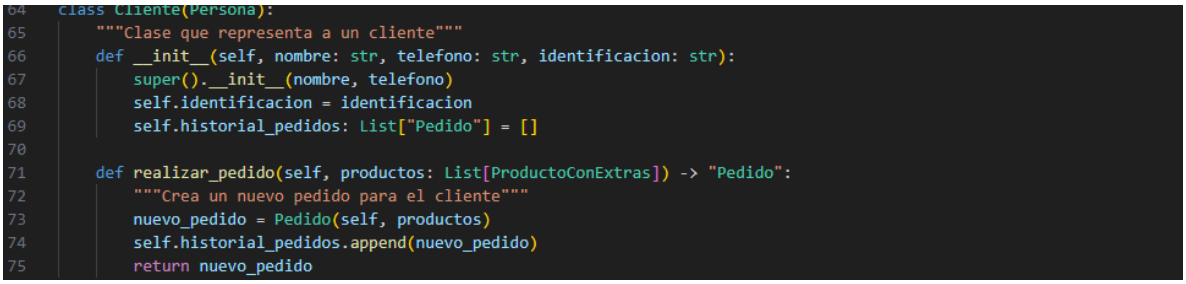
Esta es una pequeña guía donde se explican las funciones básicas que tiene el programa para que sea ocupado de la mejor manera.

**Explicación de funciones.**

Al principio después de cargar las librerías que se van a ocupar empezamos a definir nuestras clases, además de las funciones que tiene cada una, un ejemplo son los empleados, que son los que pueden realizar mas funciones que los clientes, como lo son el agregar stock de un producto, además de agregar un nuevo producto, también ellos tienen acceso a una lista de Excel donde viene toda la información de los clientes que se han registrado.

```python
class Empleado(Persona):
    """Clase que representa a un empleado"""
    def __init__(self, nombre: str, telefono: str, puesto: str, usuario: str, contrasena: str):
        super().__init__(nombre, telefono)
        self.puesto = puesto
        self.usuario = usuario
        self.contrasena = contrasena

    def procesar_pedido(self, pedido: "Pedido") -> None:
        """Cambia el estado del pedido a 'En preparación'"""
        pedido.estado = "En preparación"

    def entregar_pedido(self, pedido: "Pedido") -> None:
        """Cambia el estado del pedido a 'Entregado'"""
        pedido.estado = "Entregado"

    def actualizar_inventario(self, producto: "Producto", cantidad: int) -> None:
        """Actualiza el stock de un producto"""
        producto.stock += cantidad

```

Aquí podemos que ellos pueden procesar pedidos, además de actualizar y entregar pedidos, mientras que el cliente únicamente puede ordenar.



```python
class Cliente(Persona):
    """Clase que representa a un cliente"""
    def __init__(self, nombre: str, telefono: str, identificacion: str):
        super().__init__(nombre, telefono)
        self.identificacion = identificacion
        self.historial_pedidos: List["Pedido"] = []

    def realizar_pedido(self, productos: List[ProductoConExtras]) -> "Pedido":
        """Crea un nuevo pedido para el cliente"""
        nuevo_pedido = Pedido(self, productos)
        self.historial_pedidos.append(nuevo_pedido)
        return nuevo_pedido
```

Como vemos, solo puede realizar pedido.

```python
class Pedido:
    """Clase que representa un pedido"""
    contador_pedidos = 0

    def __init__(self, cliente: Cliente, productos: List[ProductoConExtras]):
        Pedido.contador_pedidos += 1
        self.numero = Pedido.contador_pedidos
        self.cliente = cliente
        self.productos = productos.copy()
        self.fecha = datetime.datetime.now()
        self.estado = "Nuevo"
        self.total = self.calcular_total()

    def calcular_total(self) -> float:
        """Calcula el total del pedido"""
        return sum(item.precio_total for item in self.productos)

    def agregar_producto(self, producto: ProductoConExtras) -> None:
        """Agrega un producto al pedido"""
        self.productos.append(producto)
        self.total = self.calcular_total()

    def eliminar_producto(self, producto: ProductoConExtras) -> bool:
        """Elimina un producto del pedido"""
        if producto in self.productos:
            self.productos.remove(producto)
            self.total = self.calcular_total()
            return True
        return False

    def actualizar_estado(self, nuevo_estado: str) -> None:
        """Actualiza el estado del pedido"""
```

Avanzamos a esta parte donde aquí la clase pedido es la que va hacia al cliente, pues es la que va a ayudar a calcular el total, agregar producto o eliminarlo de su pedido que se esté realizando, además de que podrá colocar si el pedido ya fue entregado.

```python
class SistemaPedidos:
    """Clase principal del sistema de pedidos"""
    DATA_FILE = "cafeteria_data.pkl"

    def __init__(self):
        self.inventario = Inventario()
        self.pedidos: List[Pedido] = []
        self.clientes: Dict[str, Cliente] = {}
        self.empleados: Dict[str, Empleado] = {}
        self.cargar_datos()

    def inicializar_datos_demo(self) -> None:
        """Inicializa datos de demostración"""
        # Bebidas
        bebidas = [
            Bebida("B001", "Café Americano", 2.50, 50, "Mediano", "☕"),
            Bebida("B002", "Cappuccino", 3.50, 30, "Mediano", "☕"),
            Bebida("B003", "Chocolate Caliente", 3.00, 25, "Mediano", "☕"),
            Bebida("B004", "Té Verde", 2.00, 40, "Mediano", "🍵"),
            Bebida("B005", "Smoothie de Frutas", 4.00, 20, "Grande", "🥤"),
            Bebida("B006", "Mocha", 3.75, 35, "Mediano", "☕"),
            Bebida("B007", "Latte", 3.25, 30, "Grande", "☕"),
            Bebida("B008", "Té de Manzanilla", 2.25, 25, "Mediano", "🍵"),
            Bebida("B009", "Jugo Natural", 3.50, 20, "Grande", "🧃"),
            Bebida("B010", "Frappé de Vainilla", 4.50, 15, "Grande", "🥤")
        ]

        # Postres
        postres = [
```

Ahora pasamos acá, donde esta nuestro menú principal, es decir, el que “de por si esta”, desde acá podemos agregar mas cosas o no, claro debe de ir en orden para

que no haya problemas, en este caso va (código, nombre, precio, stock, tamaño, y un pequeño emoji que lo representa) esto pasa lo mismo con los postres que más adelante se pondrá una imagen.

```
# Postres
postres = [
    Postre("P001", "Croissant", 2.00, 20, ["Harina", "Mantequilla", "Azúcar"], "🥐"),
    Postre("P002", "Donut", 1.50, 15, ["Harina", "Azúcar", "Chocolate"], "🍩"),
    Postre("P003", "Cheesecake", 3.50, 10, ["Queso crema", "Galleta", "Azúcar"], "🍰"),
    Postre("P004", "Brownie", 2.50, 12, ["Chocolate", "Nueces", "Harina"], "🍫"),
    Postre("P005", "Muffin de Arándanos", 2.75, 18, ["Harina", "Arándanos", "Azúcar"], "🧁"),
    Postre("P006", "Tarta de Manzana", 4.00, 8, ["Manzana", "Masa", "Canela"], "🍎"),
    Postre("P007", "Galletas de Chocolate", 1.75, 25, ["Harina", "Chocolate", "Mantequilla"], "🍪"),
    Postre("P008", "Flan", 3.00, 12, ["Huevo", "Leche", "Azúcar"], "🍮"),
    Postre("P009", "Tiramisú", 4.50, 10, ["Café", "Queso mascarpone", "Bizcochos"], "🍰"),
    Postre("P010", "Profiteroles", 3.75, 8, ["Crema", "Masa", "Chocolate"], "🧁")
]

# Agregar productos al inventario
for bebida in bebidas:
    self.inventario.agregar_producto(bebida)

for postre in postres:
    self.inventario.agregar_producto(postre)

# Agregar empleados
empleados = [
    Empleado("Amanda Sanchez", "555-1234", "Barista", "amanda", "amanda"),
    Empleado("Carlos Castillo", "555-5678", "Cajero", "carlos", "carlos"),
    Empleado("Guadalupe Mino", "555-0000", "Administrador", "admin", "admin")
]
```

seguimos en esta misma clase donde aquí se puede ver los postres además de que ya se puede agregar productos al inventario, pero como se sabe, no cualquiera puede agregar productos, es por eso por lo que acá definimos nuestros empleados, además de que sus contraseñas y usuarios para que no cualquiera pueda entrar al sistema.

```python
    def validar_empleado(self, usuario: str, contrasena: str) -> Optional[Empleado]:
        """Valida las credenciales de un empleado"""
        if usuario in self.empleados and self.empleados[usuario].contrasena == contrasena:
            return self.empleados[usuario]
        return None

    def registrar_cliente(self, nombre: str, telefono: str, identificacion: str) -> Cliente:
        """Registra un nuevo cliente"""
        nuevo_cliente = Cliente(nombre, telefono, identificacion)
        self.clientes[identificacion] = nuevo_cliente
        self.guardar_datos()
        return nuevo_cliente

    def buscar_cliente(self, identificacion: str) -> Optional[Cliente]:
        """Busca un cliente por su identificación"""
        return self.clientes.get(identificacion)

    def crear_pedido(self, cliente_id: str, productos: List[ProductoConExtras]) -> Optional[Pedido]:
        """Crea un nuevo pedido"""
        cliente = self.buscar_cliente(cliente_id)
        if not cliente:
            return None

        # Verificar stock antes de crear el pedido
        for item in productos:
            producto = self.inventario.obtener_producto(item.producto.codigo)
            if not producto or producto.stock < item.cantidad:
                messagebox.showwarning("Error", f"No hay suficiente stock de {item.producto.nombre}")
                return None
```

Aquí seguimos donde podemos ver funciones que validan los empleados con ayuda de los if, también el registrar clientes donde se les pide ciertos datos. A los clientes que ya están registrados se les busca en el sistema, y estos pueden crear un pedido, pero también se debe verificar que haya suficiente stock, es la razón que por eso el for esta mas dentro porque pertenece a def crear_pedido

```python
class SistemaPedidos:
    def crear_pedido(self, cliente_id: str, productos: List[ProductoConExtras]) -> Optional[Pedido]:
        for item in productos:
            producto = self.inventario.obtener_producto(item.producto.codigo)
            producto.actualizar_stock(-item.cantidad)

        pedido = cliente.realizar_pedido(productos)
        self.pedidos.append(pedido)
        self.guardar_datos()
        return pedido

    def listar_pedidos(self, estado: Optional[str] = None) -> List[Pedido]:
        """Lista los pedidos según su estado"""
        if estado:
            return [p for p in self.pedidos if p.estado == estado]
        return self.pedidos

    def modificar_pedido(self, numero_pedido: int, accion: str, producto: ProductoConExtras = None) -> bool:
        """Modifica un pedido existente"""
        for pedido in self.pedidos:
            if pedido.numero == numero_pedido and pedido.estado == "Nuevo":
                if accion == "agregar":
                    prod_base = self.inventario.obtener_producto(producto.producto.codigo)
                    if prod_base and prod_base.stock >= producto.cantidad:
                        pedido.agregar_producto(producto)
                        prod_base.actualizar_stock(-producto.cantidad)
                        self.guardar_datos()
                        return True
                elif accion == "eliminar":
                    for p in pedido.productos:
                        if p.producto.codigo == producto.producto.codigo:
                            if pedido.eliminar_producto(p):
```

Aquí vemos el crear pedido, pues es lo principal para cualquier venta, el listar pedidos nos ayuda a tener un reporte de ventas, además de dar seguimiento al estado en el que se encuentra el pedido, finalmente el modificar pedido ayuda a que se de flexibilidad a modificar el pedido, haciendo que se pueda cambiar y así se pueda dar una mejor atención al cliente.

```python
def procesar_pedido(self, numero_pedido: int, empleado_usuario: str) -> bool:
    """Cambia el estado del pedido a 'En preparación'"""
    empleado = self.empleados.get(empleado_usuario)
    if not empleado:
        return False

    for pedido in self.pedidos:
        if pedido.numero == numero_pedido and pedido.estado == "Nuevo":
            proceso = ProcesoPedido(pedido, empleado)
            proceso.iniciar_proceso()
            self.guardar_datos()
            return True
    return False

def entregar_pedido(self, numero_pedido: int, empleado_usuario: str) -> bool:
    """Marca el pedido como entregado"""
    empleado = self.empleados.get(empleado_usuario)
    if not empleado:
        return False

    for pedido in self.pedidos:
        if pedido.numero == numero_pedido and pedido.estado == "En preparación":
            proceso = ProcesoEntrega(pedido, empleado)
            proceso.entregar()
            self.guardar_datos()
            return True
    return False
```

Continuamos con esta parte donde aquí como dice el if, si no es empleado no lo puede realizar y es el cambiar el estado en el que se encuentra el pedido, así como marcar el pedido como entregado.

```python
    def generar_reporte_ventas(self) -> Dict:
        """Genera un reporte de ventas"""
        total_ventas = sum(p.total for p in self.pedidos if p.estado == "Entregado")
        productos_vendidos = {}
        for pedido in self.pedidos:
            if pedido.estado == "Entregado":
                for item in pedido.productos:
                    if item.producto.codigo in productos_vendidos:
                        productos_vendidos[item.producto.codigo] += item.cantidad
                    else:
                        productos_vendidos[item.producto.codigo] = item.cantidad

        return {
            "total_ventas": total_ventas,
            "productos_vendidos": productos_vendidos,
            "pedidos_completados": len([p for p in self.pedidos if p.estado == "Entregado"])
        }

    def agregar_empleado(self, nombre: str, telefono: str, puesto: str, usuario: str, contrasena: str) -> bool:
        """Agrega un nuevo empleado"""
        if usuario in self.empleados:
            return False

        nuevo_empleado = Empleado(nombre, telefono, puesto, usuario, contrasena)
        self.empleados[usuario] = nuevo_empleado
        self.guardar_datos()
        return True
```

El agregar empleado ayuda a que se agregue un nuevo empleado evitando que se dupliquen, mientras que el generar reporte de ventas, cuenta el total de pedidos que están en estado de “Entregado”.

```python
def mostrar_panel_empleado(self):
    """Muestra el panel de control para empleados"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    # Título
    ttk.Label(
        main_frame,
        text=f"🧍 Panel de Empleado - {self.empleado_actual.nombre}",
        style="Title.TLabel"
    ).pack(pady=10)

    # Información del empleado
    info_frame = ttk.Frame(main_frame, padding=10, relief=tk.RAISED, borderwidth=1)
    info_frame.pack(pady=10, fill=tk.X)

    ttk.Label(
        info_frame,
        text=f"Puesto: {self.empleado_actual.puesto}",
        font=('Helvetica', 12)
    ).pack(anchor=tk.W)

    ttk.Label(
        info_frame,
        text=f"Teléfono: {self.empleado_actual.telefono}",
        font=('Helvetica', 12)
    ).pack(anchor=tk.W)
```

Ahora pasamos a mostrar el panel de empleado, como lo es su información. Que

vienen en etiquetas donde viene el puesto y el teléfono, también se especifica el tamaño de la letra y el tipo de letra.

```
# Botones de acciones
btn_frame = ttk.Frame(main_frame)
btn_frame.pack(pady=20)

ttk.Button(
    btn_frame,
    text="➕ Agregar Producto",
    command=self.mostrar_agregar_producto,
    style="Primary.TButton"
).grid(row=0, column=0, padx=5, pady=5, sticky=tk.EW)

ttk.Button(
    btn_frame,
    text="📦 Actualizar Stock",
    command=self.mostrar_actualizar_stock,
    style="Primary.TButton"
).grid(row=0, column=1, padx=5, pady=5, sticky=tk.EW)

ttk.Button(
    btn_frame,
    text="📋 Ver Inventario",
    command=self.mostrar_inventario,
    style="Primary.TButton"
).grid(row=1, column=0, padx=5, pady=5, sticky=tk.EW)

ttk.Button(
    btn_frame,
    text="📊 Reporte de Ventas",
    command=self.mostrar_reporte_ventas,
```

Seguimos con el panel de empleados pero ahora pasamos con los botones, donde como decíamos anteriormente, estos tienen unas funciones únicas, y se puede ver mas o menos de que va en donde dice text. Van desde agregar producto hasta descargar reporte de ventas o el archivo Excel con los clientes.

```
    ttk.Button(
        btn_frame,
        text="📝 Gestionar Pedidos",
        command=self.mostrar_gestion_pedidos,
        style="Primary.TButton"
    ).grid(row=2, column=0, padx=5, pady=5, sticky=tk.EW)

    # Botón para exportar clientes
    ttk.Button(
        btn_frame,
        text="📄 Exportar Clientes",
        command=self.exportar_clientes_excel,
        style="Primary.TButton"
    ).grid(row=2, column=1, padx=5, pady=5, sticky=tk.EW)

    # Botón cerrar sesión
    ttk.Button(
        main_frame,
        text="🚪 Cerrar Sesión",
        command=self.cerrar_sesion_empleado,
        style="Secondary.TButton"
    ).pack(side=tk.BOTTOM, pady=10)
```

Aquí se puede ver los botones faltantes.

```python
class InterfazCafeteria:
    def __init__(self, root):
        self.root = root
        self.configurar_estilos()  # Primero configurar estilos
        self.mostrar_pantalla_inicio()  # Luego mostrar pantalla

    def configurar_estilos(self):
        """Configura los estilos visuales de la aplicación"""
        style = ttk.Style()

        # Configurar tema general
        style.theme_use('clam')

        # Configurar colores y fuentes
        style.configure('.',
                        background=COLORES["fondo"],
                        foreground=COLORES["texto"])

        style.configure('TFrame',
                        background=COLORES["fondo"])

        style.configure('TLabel',
                        background=COLORES["fondo"],
                        foreground=COLORES["texto"],
                        font=('Helvetica', 10))

        style.configure('TButton',
                        background=COLORES["primario"],
                        foreground='white',
                        font=('Helvetica', 10, 'bold'),
```

Ahora si pasamos a nuestra interfaz de cafetería, donde primero creamos nuestra ventana y le configuramos los estilos visuales, como son los colores que van de fondo.

```python
class InterfazCafeteria:
    def mostrar_pantalla_inicio(self):
        self.limpiar_pantalla()

        # Frame principal
        main_frame = ttk.Frame(self.root)
        main_frame.pack(expand=True, fill=tk.BOTH, padx=20, pady=20)
    """Clase para la interfaz gráfica de la cafetería"""
    def __init__(self, root):
        self.root = root
        self.sistema = SistemaPedidos()
        self.carrito: List[ProductoConExtras] = []
        self.cliente_actual = None
        self.empleado_actual = None

        # Configuración de la ventana principal
        self.root.title("☕ Sistema de Gestión de Pedidos - Cafetería Dulce Aroma")
        self.root.geometry("1000x700")
        self.root.resizable(True, True)
        self.root.configure(bg=COLORES["fondo"])

        # Configurar imagen de fondo
        self.bg_image = None
        self.bg_label = None
        self.configurar_fondo("E://Programación avanzada//fondo_inicio.jpg")

        # Configurar estilos
        self.configurar_estilos()

        # Mostrar pantalla de inicio
        self.mostrar_pantalla_inicio()
```

Aquí esta nuestra ventana principal, entonces tiene el título, la dimensión de la ventana además de que se le puso un fondo con una imagen que se descargó.

```python
def mostrar_pantalla_inicio(self):
    """Muestra la pantalla principal"""
    self.limpiar_pantalla()

    # Frame principal con degradado
    main_frame = ttk.Frame(self.root)
    main_frame.pack(expand=True, fill=tk.BOTH, padx=20, pady=20)

        # Logo y título
    logo_frame = ttk.Frame(main_frame)
    logo_frame.pack(pady=(20, 10))

    ttk.Label(logo_frame, text="☕", font=('Helvetica', 60), foreground=COLORES["primario"]).pack()
    ttk.Label(logo_frame, text="Cafetería Dulce Aroma", style="Title.TLabel").pack(pady=10)

    # Botones principales
    btn_frame = ttk.Frame(main_frame)
    btn_frame.pack(pady=30, fill=tk.X, padx=50)

    ttk.Button(btn_frame, text="Realizar Pedido", command=self.mostrar_identificacion_cliente,
               style="Primary.TButton").pack(fill=tk.X, pady=10, ipady=10)
    ttk.Button(btn_frame, text="Ver Pedidos", command=self.mostrar_lista_pedidos,
               style="Primary.TButton").pack(fill=tk.X, pady=10, ipady=10)
    ttk.Button(btn_frame, text="Soy Empleado", command=self.mostrar_login_empleado,
               style="Primary.TButton").pack(fill=tk.X, pady=10, ipady=10)
    ttk.Button(btn_frame, text="Salir", command=self.root.quit,
               style="Secondary.TButton").pack(fill=tk.X, pady=10, ipady=10)

        # Pie de página
```

Aquí seguimos con la pantalla principal pero ahora con los elementos que habrá en esta, por ejemplo, los botones que son 4 en este caso, además del logo y pie de página, esto para darle una mejor vista a nuestra página.

```python
def mostrar_lista_pedidos(self):
    """Muestra la lista de pedidos del cliente actual"""
    if not self.cliente_actual:
        messagebox.showwarning("Error", "Debe identificarse como cliente primero", parent=self.root)
        self.mostrar_identificacion_cliente()
        return

    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    # Título
    ttk.Label(
        main_frame,
        text=f"📋 Mis Pedidos - {self.cliente_actual.nombre}",
        style="Title.TLabel"
    ).pack(pady=10)

    # Frame contenedor con scroll
    container = ttk.Frame(main_frame)
    container.pack(fill=tk.BOTH, expand=True, pady=10)

    # Canvas y scrollbar
    canvas = tk.Canvas(container, bg=COLORES["fondo"], highlightthickness=0)
    scrollbar = ttk.Scrollbar(container, orient="vertical", command=canvas.yview)
    scrollable_frame = ttk.Frame(canvas)

    scrollable_frame.bind(
        "<Configure>"
```

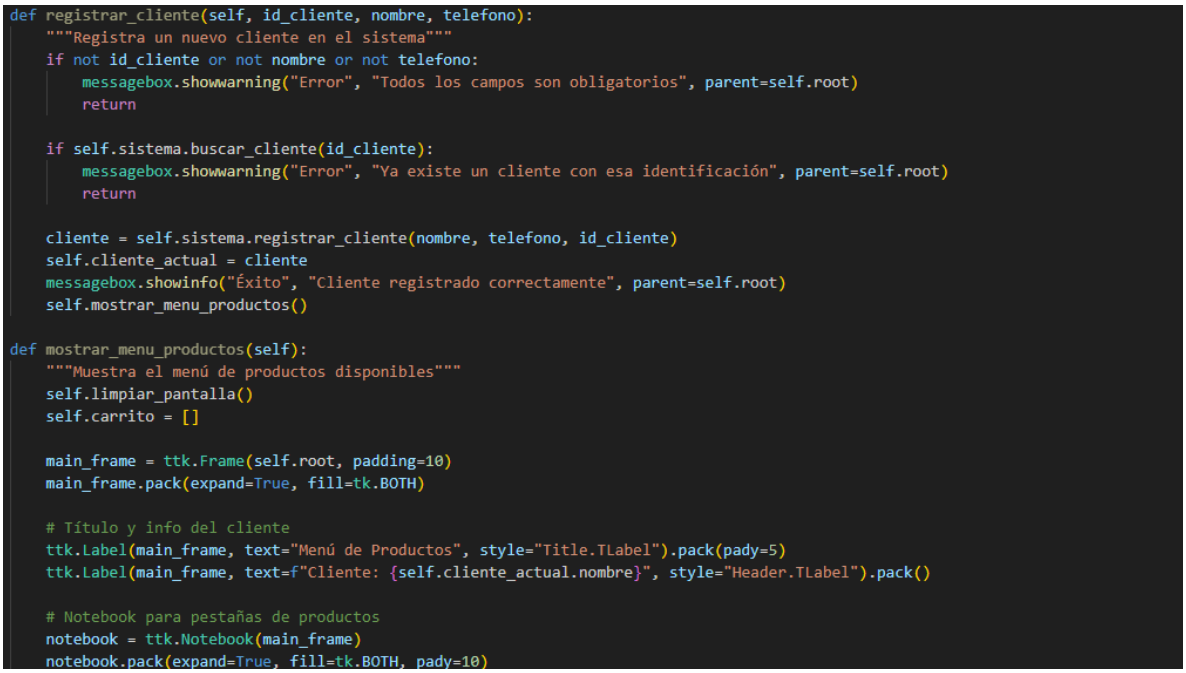
Ya dentro de lo que pase en esta interfaz es lo que poco a poco se va dando, como en este caso que al ingresar al sistema como un cliente se busca al cliente y sino le arrojara un mensaje que debe identificarse primero, además de que si pasa tendrá nuevos apartados como el “Mis pedidos”.

```python
def confirmar_recepcion(self, pedido: Pedido):
    """Confirma la recepción de un pedido entregado"""
    if messagebox.askyesno(
        "Confirmar recepción",
        f"¿Confirmas que has recibido el pedido #{pedido.numero}?",
        parent=self.root
    ):
        # En un sistema real, aquí podríamos marcar el pedido como recibido
        # Por ahora simplemente lo eliminaremos
        self.eliminar_pedido(pedido)
        messagebox.showinfo(
            "Confirmado",
            f"Pedido #{pedido.numero} marcado como recibido",
            parent=self.root
        )

def eliminar_pedido(self, pedido: Pedido):
    """Elimina un pedido del historial del cliente"""
    if messagebox.askyesno(
        "Eliminar pedido",
        f"¿Estás seguro de eliminar el pedido #{pedido.numero}?\nEsta acción no se puede deshacer.",
        parent=self.root
    ):
        # Eliminar el pedido del sistema
        if pedido in self.sistema.pedidos:
            self.sistema.pedidos.remove(pedido)

        # Eliminar el pedido del historial del cliente
        if self.cliente_actual and pedido in self.cliente_actual.historial_pedidos:
```

Ahora pasamos a estas funciones las cuales nos permiten el dar como entregado un pedido haciendo que estos se vayan eliminando, así es como se eliminan esos pedidos del sistema que ya fueron entregados además de que se borran del historial del cliente.

```python
def mostrar_registro_cliente(self):
    """Muestra el formulario de registro de cliente"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    ttk.Label(main_frame, text="Registro de Cliente", style="Title.TLabel").pack(pady=10)

    # Frame del formulario
    form_frame = ttk.Frame(main_frame, padding=20, relief=tk.RAISED, borderwidth=2)
    form_frame.pack(pady=20, ipadx=20, ipady=20)

    # Campos del formulario
    ttk.Label(form_frame, text="Identificación:", font=('Helvetica', 12)).grid(row=0, column=0, pady=10, sticky=tk.W)
    id_entry = ttk.Entry(form_frame, font=('Helvetica', 12), width=30)
    id_entry.grid(row=0, column=1, pady=10, padx=10)

    ttk.Label(form_frame, text="Nombre:", font=('Helvetica', 12)).grid(row=1, column=0, pady=10, sticky=tk.W)
    nombre_entry = ttk.Entry(form_frame, font=('Helvetica', 12), width=30)
    nombre_entry.grid(row=1, column=1, pady=10, padx=10)

    ttk.Label(form_frame, text="Teléfono:", font=('Helvetica', 12)).grid(row=2, column=0, pady=10, sticky=tk.W)
    telefono_entry = ttk.Entry(form_frame, font=('Helvetica', 12), width=30)
    telefono_entry.grid(row=2, column=1, pady=10, padx=10)

    # Botón registrar
    ttk.Button(form_frame, text="Registrar",
               command=lambda: self.registrar_cliente(
                   id_entry.get(), nombre_entry.get(), telefono_entry.get()
```

En caso de que el usuario no este registrado se le mencionara si gusta registrarse, si acepta que sí, tendrá que llenar este pequeño formulario, donde se le pedirá, identificación, nombre, teléfono, además de que al final existe el botón de registrar.



```
def registrar_cliente(self, id_cliente, nombre, telefono):
    """Registra un nuevo cliente en el sistema"""
    if not id_cliente or not nombre or not telefono:
        messagebox.showwarning("Error", "Todos los campos son obligatorios", parent=self.root)
        return

    if self.sistema.buscar_cliente(id_cliente):
        messagebox.showwarning("Error", "Ya existe un cliente con esa identificación", parent=self.root)
        return

    cliente = self.sistema.registrar_cliente(nombre, telefono, id_cliente)
    self.cliente_actual = cliente
    messagebox.showinfo("Éxito", "Cliente registrado correctamente", parent=self.root)
    self.mostrar_menu_productos()

def mostrar_menu_productos(self):
    """Muestra el menú de productos disponibles"""
    self.limpiar_pantalla()
    self.carrito = []

    main_frame = ttk.Frame(self.root, padding=10)
    main_frame.pack(expand=True, fill=tk.BOTH)

    # Título y info del cliente
    ttk.Label(main_frame, text="Menú de Productos", style="Title.TLabel").pack(pady=5)
    ttk.Label(main_frame, text=f"Cliente: {self.cliente_actual.nombre}", style="Header.TLabel").pack()

    # Notebook para pestañas de productos
    notebook = ttk.Notebook(main_frame)
    notebook.pack(expand=True, fill=tk.BOTH, pady=10)
```

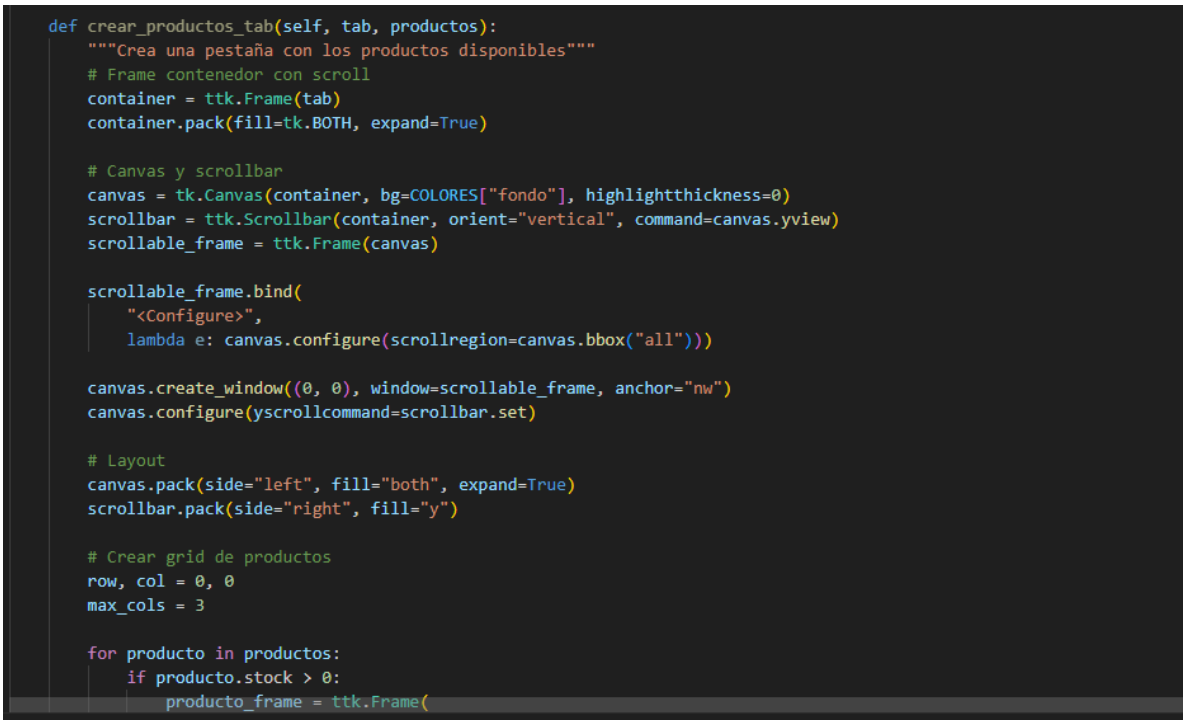
Aquí tenemos parte de dos funciones importantes, primeramente, la de registrar cliente donde en caso de que exista alguien con su misma información el sistema simplemente no dejara completar el registro arrojando un mensaje de error. La otra función nos sirve para mostrar el menú de productos de la cafetería, en donde como ya hemos visto anteriormente se mostrarán una inmensa cantidad de productos con los que cuenta la cafetería.

```python
# Pestaña Bebidas
bebidas_tab = ttk.Frame(notebook)
notebook.add(bebidas_tab, text="☕ Bebidas")
self.crear_productos_tab(bebidas_tab, self.sistema.inventario.listar_bebidas())

# Pestaña Postres
postres_tab = ttk.Frame(notebook)
notebook.add(postres_tab, text="🍰 Postres")
self.crear_productos_tab(postres_tab, self.sistema.inventario.listar_postres())

# Frame para carrito
carrito_frame = ttk.LabelFrame(main_frame, text="🛒 Carrito de Compras", padding=10)
carrito_frame.pack(fill=tk.X, pady=10)

# Listbox para el carrito con scrollbar
self.carrito_listbox = tk.Listbox(
    carrito_frame,
    height=5,
    font=('Helvetica', 10),
    selectbackground=COLORES["primario"],
    selectforeground='white'
)
self.carrito_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)

scrollbar = ttk.Scrollbar(carrito_frame, orient=tk.VERTICAL, command=self.carrito_listbox.yview)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
self.carrito_listbox.config(yscrollcommand=scrollbar.set)
```

Aquí podemos ver mas sobre esto, la pestaña de bebidas y la de postres además de nuestro carrito a donde van todos los productos que el cliente vaya comprando.

```python
def mostrar_menu_productos(self):
    self.total_label.pack(anchor=tk.E, padx=20)

    # Botones de acción
    btn_frame = ttk.Frame(main_frame)
    btn_frame.pack(pady=10, fill=tk.X, padx=20)

    ttk.Button(
        btn_frame,
        text="❌ Eliminar Seleccionado",
        command=self.eliminar_del_carrito,
        style="Secondary.TButton"
    ).pack(side=tk.LEFT, padx=5)

    ttk.Button(
        btn_frame,
        text="✅ Realizar Pedido",
        command=self.finalizar_pedido,
        style="Primary.TButton"
    ).pack(side=tk.RIGHT, padx=5)

    ttk.Button(
        btn_frame,
        text="↩️ Volver",
        command=self.mostrar_pantalla_inicio,
        style="Secondary.TButton"
    ).pack(side=tk.RIGHT, padx=5)
```

Seguimos con estos botones que nos sirven para modificar de manera rápida además de que también está la función de realizar pedido que nos ayudara a finalizar el pedido.



```
def crear_productos_tab(self, tab, productos):
    """Crea una pestaña con los productos disponibles"""
    # Frame contenedor con scroll
    container = ttk.Frame(tab)
    container.pack(fill=tk.BOTH, expand=True)

    # Canvas y scrollbar
    canvas = tk.Canvas(container, bg=COLORES["fondo"], highlightthickness=0)
    scrollbar = ttk.Scrollbar(container, orient="vertical", command=canvas.yview)
    scrollable_frame = ttk.Frame(canvas)

    scrollable_frame.bind(
        "<Configure>",
        lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

    canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
    canvas.configure(yscrollcommand=scrollbar.set)

    # Layout
    canvas.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    # Crear grid de productos
    row, col = 0, 0
    max_cols = 3

    for producto in productos:
        if producto.stock > 0:
            producto_frame = ttk.Frame(
```

Aquí tenemos una función que nos permite el crear productos, claramente nos da opciones a elegir y depende a lo que sea el producto, este tomara su forma.

```python
def crear_productos_tab(self, tab, productos):
        # "Imagen" (emoji)
        img_label = ttk.Label(
            producto_frame,
            text=producto.imagen if hasattr(producto, 'imagen') and producto.imagen else "☕" if isinstance(producto, Bebida) else "🍰",
            font=('Helvetica', 36),
            foreground=COLORES["primario"]
        )
        img_label.pack(pady=5)

        # Detalles específicos
        detalles_frame = ttk.Frame(producto_frame)
        detalles_frame.pack(fill=tk.X, pady=5)

        if isinstance(producto, Bebida):
            ttk.Label(detalles_frame, text=f"Tamaño: {producto.tamano}").pack(anchor=tk.W)
        elif isinstance(producto, Postre):
            ttk.Label(detalles_frame, text=f"Ingredientes: {producto.mostrar_ingredientes()}").pack(anchor=tk.W)

        ttk.Label(detalles_frame, text=f"💲 Precio: ${producto.precio:.2f}", font=('Helvetica', 11, 'bold')).pack(anchor=tk.W)
        ttk.Label(detalles_frame, text=f"📦 Disponible: {producto.stock}",
                  foreground=COLORES["exito"] if producto.stock > 5 else COLORES["advertencia"] if producto.stock > 0 else COLORES["error
                  font=('Helvetica', 10)).pack(anchor=tk.W)

        # Botón agregar con opciones
        self.crear_boton_agregar(producto_frame, producto)

        # Posicionamiento en el grid
        producto_frame.grid(
            row=row,
            column=col,
```

Aquí esta un ejemplo que depende a lo que seleccionemos, se le pondrá una imagen de una taza de café si es bebida o una de un pastelito si es postre.

```python
def crear_boton_agregar(self, parent, producto):
    """Crea el botón de agregar con opciones para el producto"""
    def mostrar_opciones():
        opciones_window = tk.Toplevel(self.root)
        opciones_window.title(f"Opciones para {producto.nombre}")
        opciones_window.geometry("400x300")
        opciones_window.resizable(False, False)

        # Variables para las opciones
        cantidad = tk.IntVar(value=1)
        tipo_leche = tk.StringVar(value=TIPOS_LECHE[0] if isinstance(producto, Bebida) else "")
        nivel_azucar = tk.StringVar(value=NIVELES_AZUCAR[2] if isinstance(producto, Bebida) else "")
        notas = tk.StringVar()

        # Frame principal
        main_frame = ttk.Frame(opciones_window, padding=10)
        main_frame.pack(expand=True, fill=tk.BOTH)

        # Cantidad
        ttk.Label(main_frame, text="Cantidad:").grid(row=0, column=0, sticky=tk.W, pady=5)
        ttk.Spinbox(main_frame, from_=1, to=10, textvariable=cantidad, width=5).grid(row=0, column=1, sticky=tk.W, pady=5)

        # Opciones específicas para bebidas
        if isinstance(producto, Bebida):
            # Tipo de leche
            ttk.Label(main_frame, text="Tipo de leche:").grid(row=1, column=0, sticky=tk.W, pady=5)
            ttk.Combobox(
                main_frame,
                textvariable=tipo_leche,
                values=TIPOS_LECHE,
```

Este apartado para mi es de suma importancia, pues primeramente nos abrirá otra ventana, con esas dimensiones que permitirá al cliente el poder cambiar su producto, además de editarlo en cuestión de algún tipo de leche o con cuanto nivel

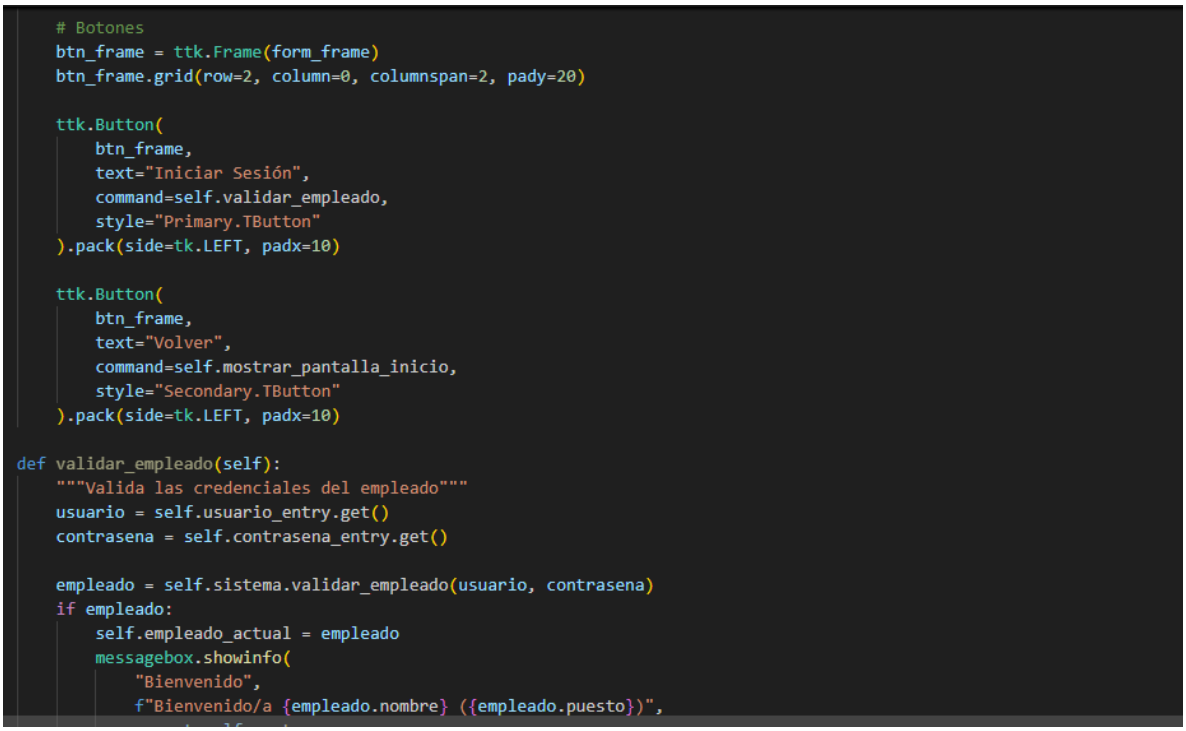
de azúcar, con esto se busca una atención eficaz y que el cliente se sienta conforme con su compra.

```
            # Nivel de azúcar
            ttk.Label(main_frame, text="Nivel de azúcar:").grid(row=2, column=0, sticky=tk.W, pady=5)
            ttk.Combobox(
                main_frame,
                textvariable=nivel_azucar,
                values=NIVELES_AZUCAR,
                state="readonly"
            ).grid(row=2, column=1, sticky=tk.W, pady=5)

        # Notas adicionales
        ttk.Label(main_frame, text="Notas adicionales:").grid(row=3, column=0, sticky=tk.W, pady=5)
        ttk.Entry(main_frame, textvariable=notas).grid(row=3, column=1, sticky=tk.W, pady=5)

        # Botones
        btn_frame = ttk.Frame(main_frame)
        btn_frame.grid(row=4, column=0, columnspan=2, pady=10)

        ttk.Button(
            btn_frame,
            text="Agregar al Carrito",
            command=lambda: self.agregar_con_opciones(
                producto,
                cantidad.get(),
                tipo_leche.get() if isinstance(producto, Bebida) else None,
                nivel_azucar.get() if isinstance(producto, Bebida) else None,
                notas.get(),
                opciones_window
            ),
```

Después de cambiar su producto aparecerá un botón de agregar al carrito con todas esas notas especiales que el usuario requiere.

```
def mostrar_login_empleado(self):
    """Muestra el formulario de inicio de sesión para empleados"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    ttk.Label(
        main_frame,
        text="🔐 Inicio de Sesión para Empleados",
        style="Title.TLabel"
    ).pack(pady=10)

    # Frame del formulario
    form_frame = ttk.Frame(
        main_frame,
        padding=20,
        relief=tk.RAISED,
        borderwidth=2
    )
    form_frame.pack(pady=20, ipadx=20, ipady=20)

    # Campos del formulario
    ttk.Label(
        form_frame,
        text="👤 Usuario:",
        font=('Helvetica', 12)
    ).grid(row=0, column=0, pady=10, sticky=tk.W)
```

Ahora cambiamos algo drástico, pues ahora nos vamos con el formulario si es que oprimimos el botón de “Soy empleado” el cual es un inicio de sesión para empleados donde se pedirá el usuario y contraseña para validar si es o no un empleado.



```
    # Botones
    btn_frame = ttk.Frame(form_frame)
    btn_frame.grid(row=2, column=0, columnspan=2, pady=20)

    ttk.Button(
        btn_frame,
        text="Iniciar Sesión",
        command=self.validar_empleado,
        style="Primary.TButton"
    ).pack(side=tk.LEFT, padx=10)

    ttk.Button(
        btn_frame,
        text="Volver",
        command=self.mostrar_pantalla_inicio,
        style="Secondary.TButton"
    ).pack(side=tk.LEFT, padx=10)

def validar_empleado(self):
    """Valida las credenciales del empleado"""
    usuario = self.usuario_entry.get()
    contrasena = self.contrasena_entry.get()

    empleado = self.sistema.validar_empleado(usuario, contrasena)
    if empleado:
        self.empleado_actual = empleado
        messagebox.showinfo(
            "Bienvenido",
            f"Bienvenido/a {empleado.nombre} ({empleado.puesto})",
```

Eso lo notamos aquí pues están los botones de validar o de volver, la siguiente función es el validar las credenciales del empleado, y con el if, ponemos que es lo que queremos que le aparezca, en este caso, un saludo.

```python
def mostrar_panel_empleado(self):
    """Muestra el panel de control para empleados"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    # Título
    ttk.Label(
        main_frame,
        text=f" Panel de Empleado - {self.empleado_actual.nombre}",
        style="Title.TLabel"
    ).pack(pady=10)

    # Información del empleado
    info_frame = ttk.Frame(main_frame, padding=10, relief=tk.RAISED, borderwidth=1)
    info_frame.pack(pady=10, fill=tk.X)

    ttk.Label(
        info_frame,
        text=f"Puesto: {self.empleado_actual.puesto}",
        font=('Helvetica', 12)
    ).pack(anchor=tk.W)

    ttk.Label(
        info_frame,
        text=f"Teléfono: {self.empleado_actual.telefono}",
        font=('Helvetica', 12)
    ).pack(anchor=tk.W)
```

Anteriormente habíamos mostrado las funciones que eran únicas para los empleados, pues ahora si es que el empleado fue validado, obtendrá acceso a una ventana donde las funciones que mencionamos anteriormente estarán disponibles además de que un cuadro de texto con información de esta persona, como su puesto, teléfono y nombre.

```python
def mostrar_gestion_pedidos(self):
    """Muestra la interfaz para gestionar pedidos (empleados)"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    # Título
    ttk.Label(
        main_frame,
        text="📝 Gestión de Pedidos",
        style="Title.TLabel"
    ).pack(pady=10)

    # Notebook para pestañas de estados
    notebook = ttk.Notebook(main_frame)
    notebook.pack(expand=True, fill=tk.BOTH, pady=10)

    # Pestaña de pedidos nuevos
    nuevos_tab = ttk.Frame(notebook)
    notebook.add(nuevos_tab, text="🆕 Nuevos")
    self.crear_lista_pedidos_empleado(nuevos_tab, "Nuevo")

    # Pestaña de pedidos en preparación
    preparacion_tab = ttk.Frame(notebook)
    notebook.add(preparacion_tab, text="⏳ En Preparación")
    self.crear_lista_pedidos_empleado(preparacion_tab, "En preparación")

    # Pestaña de pedidos entregados
```

Ahora continuamos con este apartado que es único para empleados, donde se gestionan los pedidos, es decir aquí aparecen los pedidos que los clientes han realizado y el empleado puede cambiar el estatus en el que se encuentra para que se lleve un buen control de cada pedido y así saber cual es el siguiente que hay que tomar en cuenta.

```
def mostrar_agregar_producto(self):
    """Muestra el formulario para agregar un nuevo producto"""
    self.limpiar_pantalla()

    main_frame = ttk.Frame(self.root, padding=20)
    main_frame.pack(expand=True, fill=tk.BOTH)

    ttk.Label(
        main_frame,
        text="➕ Agregar Producto",
        style="Title.TLabel"
    ).pack(pady=10)

    # Frame del formulario
    form_frame = ttk.Frame(
        main_frame,
        padding=20,
        relief=tk.RAISED,
        borderwidth=2
    )
    form_frame.pack(pady=20, ipadx=20, ipady=20)

    # Campos del formulario
    ttk.Label(
        form_frame,
        text="🆔 Código:",
        font=('Helvetica', 12)
    ).grid(row=0, column=0, pady=10, sticky=tk.W)
```

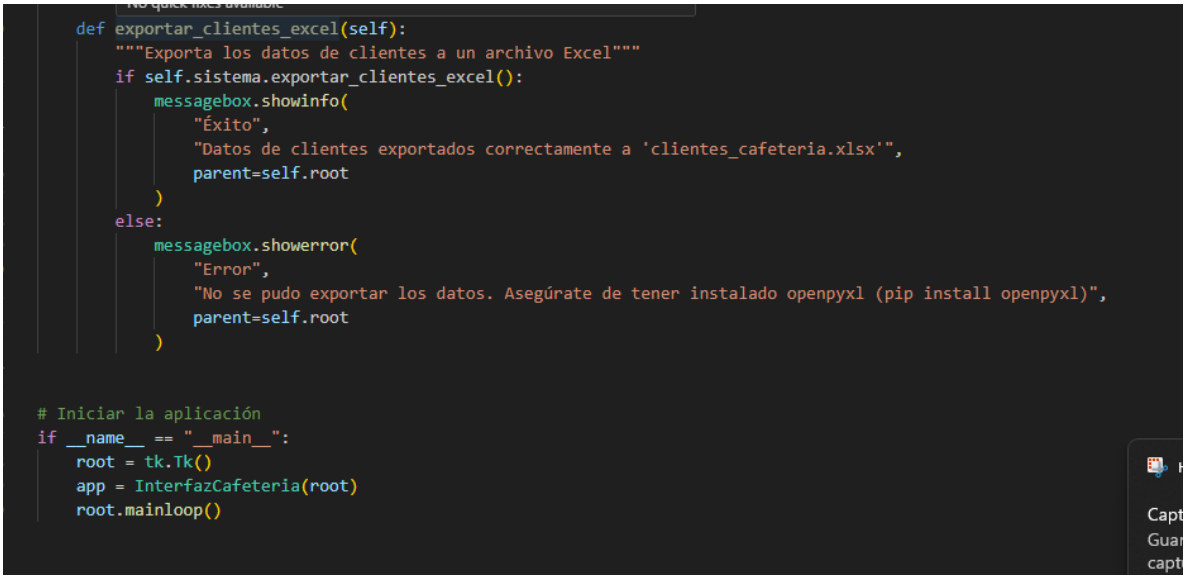
Aquí se nos muestra la función de agregar un nuevo producto, el cual es un tipo cuestionario donde nos ide información como el código para que se pueda agregar, además de información necesaria para dar de alta el producto.

```
    ttk.Label(
        form_frame,
        text="👄 Nombre:",
        font=('Helvetica', 12)
    ).grid(row=1, column=0, pady=10, sticky=tk.W)

    nombre_entry = ttk.Entry(
        form_frame,
        font=('Helvetica', 12),
        width=30
    )
    nombre_entry.grid(row=1, column=1, pady=10, padx=10)

    ttk.Label(
        form_frame,
        text="💲 Precio:",
        font=('Helvetica', 12)
    ).grid(row=2, column=0, pady=10, sticky=tk.W)

    precio_entry = ttk.Entry(
        form_frame,
        font=('Helvetica', 12),
        width=30
    )
    precio_entry.grid(row=2, column=1, pady=10, padx=10)

    ttk.Label(
        form_frame,
        text="📦 Stock:",
```

El nombre, stock y precio son algunos de los datos que se piden para así poder dar de alta el producto y este pueda comenzar a comercializarse.

```
def mostrar_actualizar_stock(self):

    # Lista de productos
    for producto in self.sistema.inventario.listar_productos():
        producto_frame = ttk.Frame(
            scrollable_frame,
            borderwidth=1,
            relief="solid",
            padding=10
        )
        producto_frame.pack(fill=tk.X, pady=5, padx=5)

        # Información del producto
        ttk.Label(
            producto_frame,
            text=f"{producto.nombre} (Código: {producto.codigo})",
            font=('Helvetica', 11, 'bold')
        ).pack(anchor=tk.W)

        ttk.Label(
            producto_frame,
            text=f"Stock actual: {producto.stock}",
            font=('Helvetica', 10)
        ).pack(anchor=tk.W)

        # Botón actualizar
        ttk.Button(
            producto_frame,
            text="Actualizar Stock",
            command=lambda p=producto: self.actualizar_stock_producto(p),
            style="Primary.TButton"
```

Esta función nos ayuda a agregar stock a productos que ya existen, así que es necesario mencionar el stock actual y después el actualizar stock.



```
    def exportar_clientes_excel(self):
        """Exporta los datos de clientes a un archivo Excel"""
        if self.sistema.exportar_clientes_excel():
            messagebox.showinfo(
                "Éxito",
                "Datos de clientes exportados correctamente a 'clientes_cafeteria.xlsx'",
                parent=self.root
            )
        else:
            messagebox.showerror(
                "Error",
                "No se pudo exportar los datos. Asegúrate de tener instalado openpyxl (pip install openpyxl)",
                parent=self.root
            )


# Iniciar la aplicación
if __name__ == "__main__":
    root = tk.Tk()
    app = InterfazCafeteria(root)
    root.mainloop()
```

Finalmente, otra función que creo es necesario mencionar es la del exportar la información de los clientes a un Excel, donde se guardaran y en caso de éxito, arrojara el mensaje y en caso de que no, marcara un error, finalmente se inicia la aplicación y así terminamos nuestra cafetería.

Ya que se han explicado algunas funciones pasaremos a explicar de forma mas general a quienes va dirigido el sistema y que se busca solucionar.

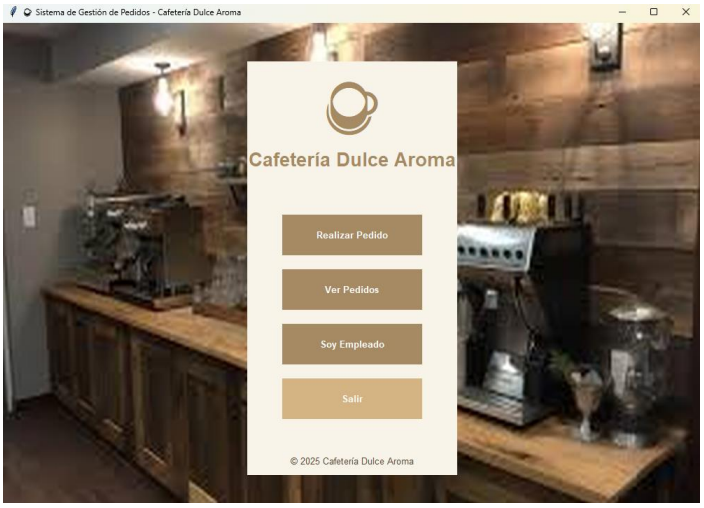
**¿Cómo funciona el sistema de cafetería dulce aroma?**

El sistema de cafetería principalmente busca que esta cafetería tenga un buen control sobre sus ventas, además de que la información de sus clientes sea estrictamente usada para control por parte de la cafetería por lo que se requiere que el sistema solo permita a empleados o administradores ingresar al total del sistema, esto con la ayuda de un usuario y una contraseña. De esta forma habrá un control sobre quienes acceden a esta información además de otras funciones como se han explicado anteriormente como el agregar un producto o cambiar el stock, además de ser los que saben los pedidos de cada cliente, haciendo que estén ordenados y se lleve un buen control.

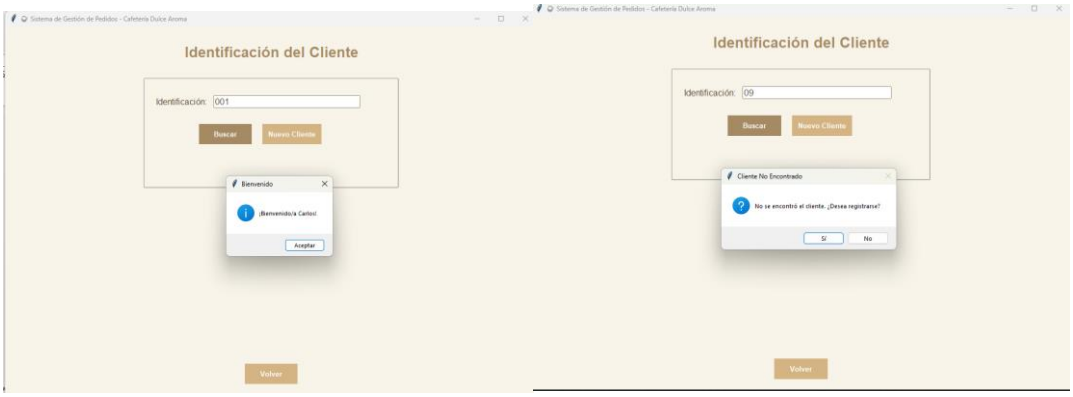
Ahora, el cliente también debe sentirse cómodo con el sistema, es por eso que este sistema es amigable con ellos, pues solo para registrarse se piden los datos necesarios, además de que se da un número de identificación que ellos mismos proporcionan, esto para que nadie haga mal uso de su cuenta, como el pedir sin su consentimiento. Además de que se intentó hacer el menú de una forma agradable, es decir que se entendiera, también con el implemento de que el usuario pueda hacer algunas modificaciones con su pedido como el tipo de leche o el nivel de azúcar, se busca que el cliente quede satisfecho con su compra. Esto con la implementación de varios botones y mensajes para que se de a entender que es lo que va pasando a lo largo que se crea un pedido.

Después de explicar parte del sistema, se hará una guía para que cualquiera ocupe este sistema de forma adecuada.

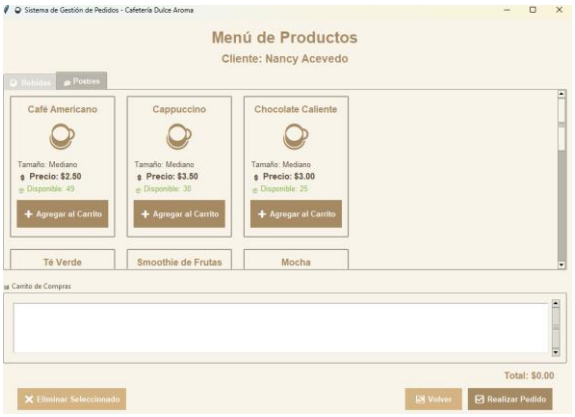
## Guía de uso. (Cliente)



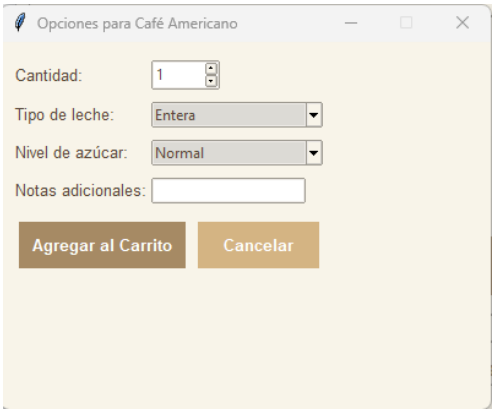

Aquí observamos la interfaz principal, para ingresar a ver pedidos, es necesario el iniciar sesión con nuestro id, es por eso que ingresaremos a realizar pedido donde se nos pedirá nuestro id o en caso de no tener, la opción de poder registrarnos.



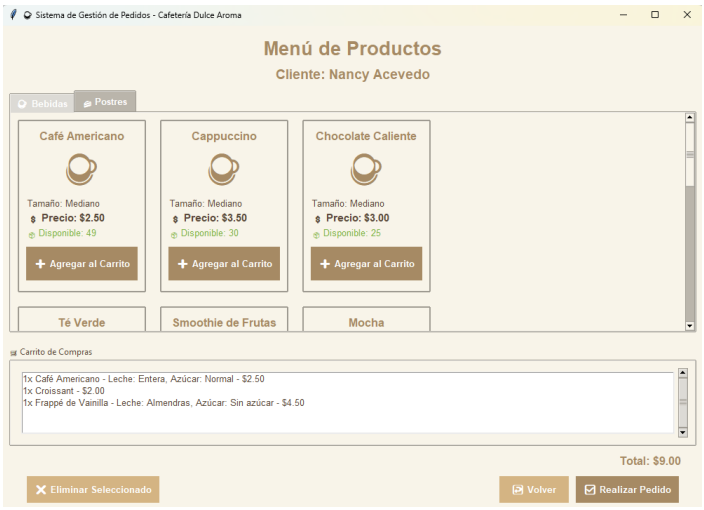

Si damos en registrarnos, nos pedirá información esencial para darnos de alta en el sistema, es importante recordar que id colocamos pues es la clave para ingresar.

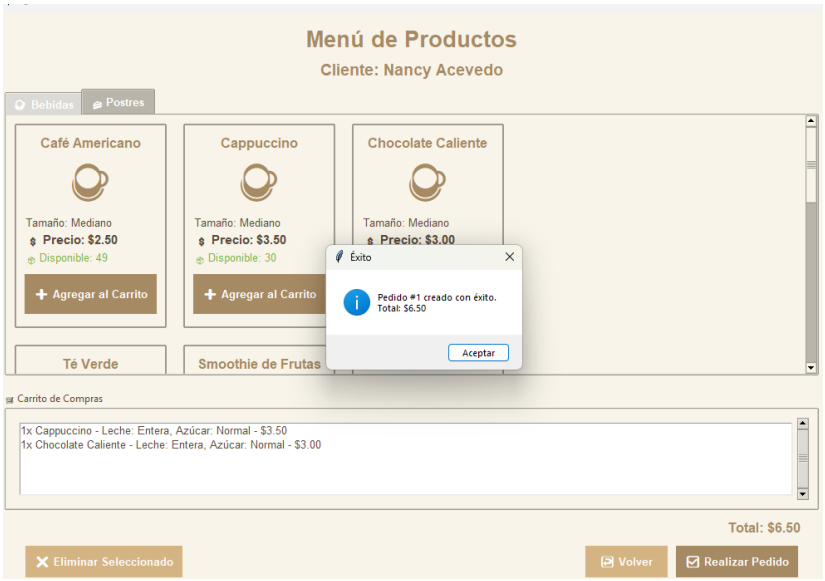

Después de contestar el cuestionario, directamente nos mandara al menú donde podremos realizar nuestro pedido, como deseemos, recordemos que hay una pestaña de postres y una de bebidas, elegimos lo que se nos antoja y lo veremos poco a poco en el cuadro de abajo.



Como sabemos, algunos productos son personalizables, así que los podremos personalizar para así obtener la bebida que queremos, después de hacer los cambios necesarios damos en agregar al carrito.




Como se observa se ha creado un pedido, si ya queremos finalizarlo, tenemos la opción de realizar pedido, o la opción de eliminar lo seleccionado que es para eliminar un producto que seleccionemos de nuestro carrito de compras, o el botón de volver que nos regresara al inicio sin confirmar ningún pedido.




Al realizar pedido, podremos ver un mensaje de que nuestro pedido fue realizado con éxito, finalizando este pedido. Esto nos llevara a la ultima pestaña que es donde nos dirá como va nuestro pedido, además de un resumen de todo.

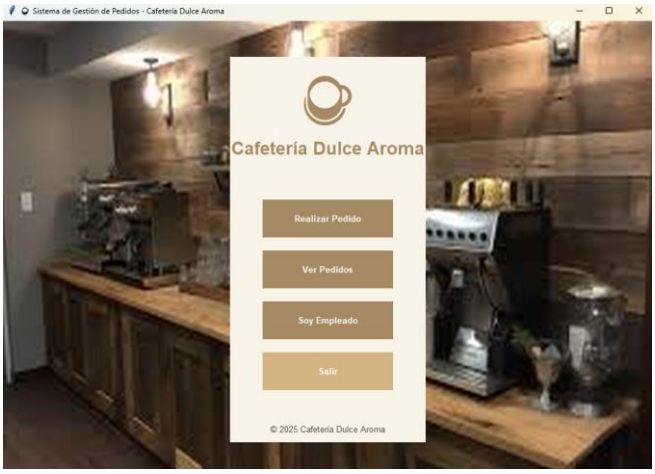

Abajo aparecen dos botones si es que queremos realizar otro pedido es volver al menú o si queremos volver al inicio para así salir de nuestra cuenta y volver a la ventana principal.

Listo, ya puedes ocupar la página de cafetería Dulce Aroma, a disfrutar.


Si oprimimos ahora si en el botón de la ventana principal de ver pedidos podremos ver nuestros pedidos que se han hecho.

Si queremos salir de nuestra cuenta solo cerramos el sistema con el botón de hasta debajo que se encuentra en la ventana principal “salir” así finalizando de ocupar este sistema.

## Guía de uso (Empleado)



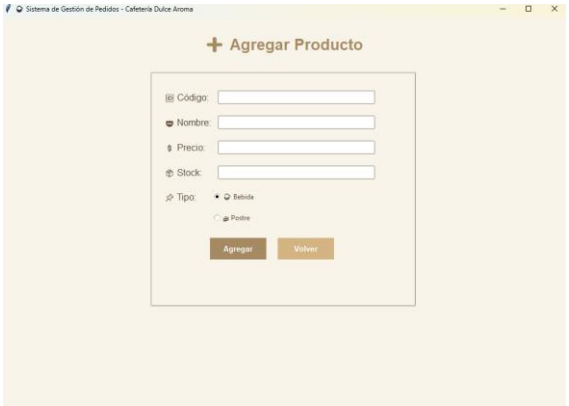

La ventana principal es la misma para todos, con la diferencia que veremos cómo funciona el botón de soy empleado, después de dar ahí, nos aparecerá la ventana donde nos pedirá el usuario y la contraseña, misma que en esta empresa son las siguientes.


```
# Agregar empleados
empleados = [
    Empleado("Amanda Sanchez", "555-1234", "Barista", "amanda", "amanda"),
    Empleado("Carlos Castillo", "555-5678", "Cajero", "carlos", "carlos"),
    Empleado("Guadalupe Mino", "555-0000", "Administrador", "admin", "admin")
```

El usuario es la penúltima palabra y la contraseña la última. Una vez ingresando nos aparecerá esta ventana.

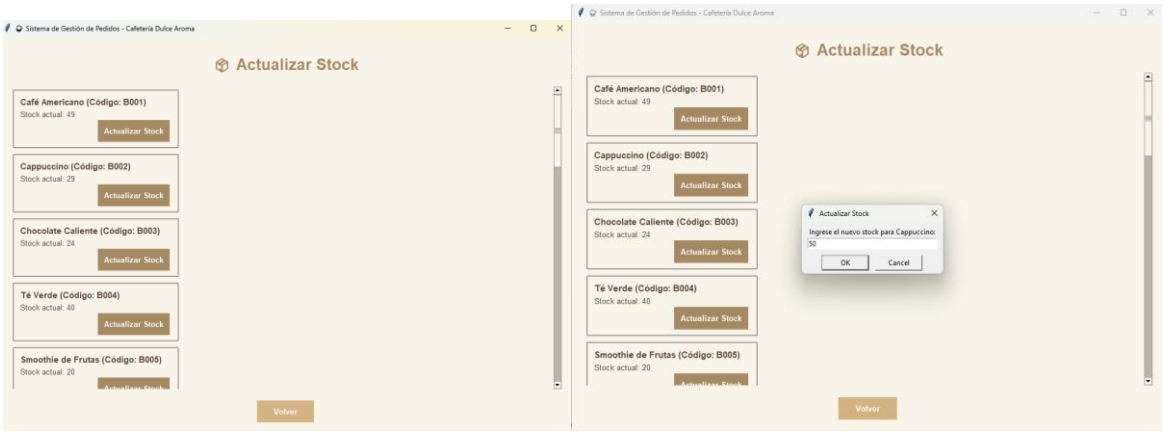

En esta ocasión entraremos como Guadalupe Mino, quien es administrador, vemos que nos aparece información necesaria como puesto y teléfono, además de las funciones que en el pasado explicamos. Ahora explicaremos de forma rápida en que consiste cada uno

*Agregar producto*



Nos dará este cuestionario donde se nos pedirá información para crear un nuevo producto, al dar en agregar este será un nuevo producto que los clientes podrán disfrutar.

*Actualizar stock*




Con esta función podrás actualizar el stock disponible de un producto existente, solo es necesario poner cual es el nuevo stock y este se cambiará, dando así, una mayor existencia de productos lideres buscando que todos lo puedan probar.

*Ver inventario*

Aquí podremos ver el menú completo, además con características esenciales de cada producto, esto para tener un mejor control de que es lo que se vende en la cafetería.

*Gestión de pedidos*


Aquí podremos ver que ha pasado con las ventas, además de que de aquí tenemos el control de cada pedido y poder cambiar el estatus del pedido, para después poder marcarlo como entregado.

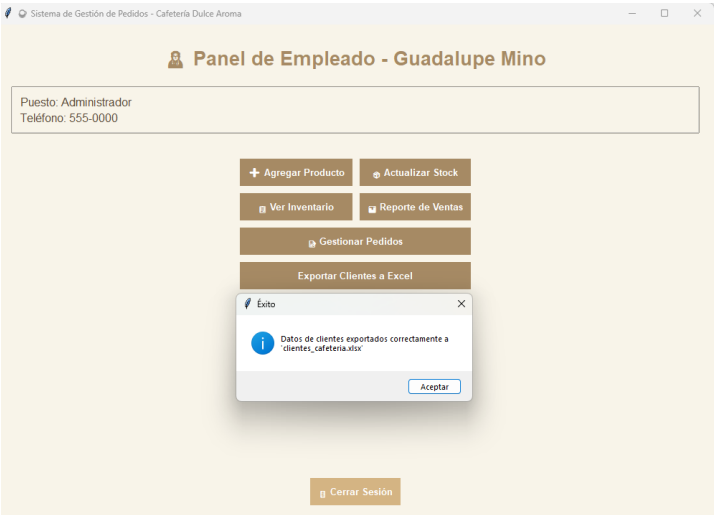

Sistema de Gestión de Pedidos - Cafetería Dulce Aroma
Gestión de Pedidos
Nuevos
Entregados
Pedido #1 - Cliente: Carlos
Fecha: 21/04/2025 15:54
• 1x Café Americano - Leche: Entera, Azúcar: Normal - $2.50
Total: $2.50
Marcar como Entregado
Volver


Sistema de Gestión de Pedidos - Cafetería Dulce Aroma
Gestión de Pedidos
Nuevos
En Preparación
Pedido #1 - Cliente: Carlos
Fecha: 21/04/2025 15:54
• 1x Café Americano - Leche: Entera, Azúcar: Normal - $2.50
Total: $2.50
Volver

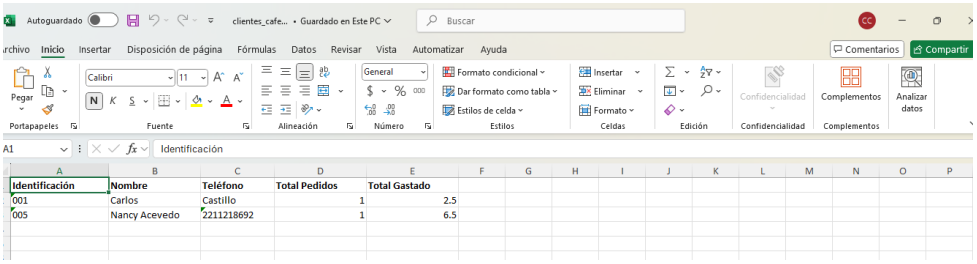
*Reporte de ventas*


Aquí podremos ver estadísticas de los pedidos realizados en general, además de los productos mas vendidos, con esto se busca que haya una mejor atención a la demanda de los clientes en ciertos productos que llamaremos “ganadores”.

*Exportar clientes a Excel*




Finalmente, este botón nos servirá para que todos los clientes que se han registrado sean mandados a un archivo Excel, donde podremos ver su información y así ocuparla para conteos necesarios o estadísticas, sin dañar la privacidad de los clientes.




Listo, ahora ya puedes manejar la cafetería, a disfrutar y a endulzar la vida de los demás…