**TRABAJO PRÁCTICO INTEGRADOR**

**Materia**: Algoritmos y estructuras de datos I

**Integrantes**: Cao Ivan, Hanisch Andres, Sica Franco y Olazar Maximo

**Nombre del Equipo**: El Koto



**Propuesta**: El proyecto consiste en la creación de un Software para un Supermercado, que abarque distintas áreas, enfocado para el uso en Caja como para uso Administrativo en cuanto a diversas funciones.

El mismo contará con un menú interactivo para que el usuario pueda:

* Consultar información de productos (precio, promociones, stock, descripción, ubicación física)
* Alta, baja y modificación de productos.
* Calculadora integrada para realizar las diferentes operaciones en caja. A su vez, estará conectada con la base de datos de los productos, por lo que disminuirá su stock, así como devolverá el total a pagar, y el cambio a entregar.
* Creación de estadísticas generales, como cantidad de ventas, facturación, productos más vendidos.

**Funcionalidades Implementadas:**

1. **Menú Principal**:
   * El programa presenta un menú interactivo que permite al usuario navegar por las opciones.
   * Opciones disponibles:
     + Consultar información de productos.
     + Alta, baja y modificación (ABM) de productos.
     + Funciones de caja.
     + Generación de estadísticas.
     + Salir del programa.
2. **Consulta de Información de Productos**:
   * Permite visualizar los productos almacenados en un archivo JSON.
   * Muestra datos como precio, promociones, stock y ubicación física del producto.
3. **ABM de Productos (Alta, Baja, Modificación)**:
   * **Agregar Productos**: Se pueden añadir nuevos productos al sistema con todos los detalles requeridos.
   * **Modificar Productos**: Es posible editar los detalles de un producto ya existente.
   * **Eliminar Productos**: Los productos se pueden eliminar del sistema de manera permanente.
4. **Manejo de Archivos JSON**:
   * Los datos de los productos se almacenan en un archivo JSON.
   * Para esto, se importo la librería json de Python, la cual permite leer, modificar y guardar la información de los productos en el archivo json.
   * Se incluyen funciones para gestionar los archivos JSON, que almacenan todos los productos con sus detalles.
5. **Caja**:
   * Conexión con la base de datos de productos para realizar operaciones de venta.
   * Reducción automática de stock tras cada operación y cálculo del cambio.
6. **Generación de Estadísticas**:
   * Herramientas para crear reportes sobre las ventas, facturación y productos más vendidos.
7. **Salida del Programa**:
   * Una opción que muestra un banner de despedida y finaliza la ejecución.

**Guía de usuario:** Explicación de las funciones (ordenadas) y bibliotecas usadas en el Software.

**Bibliotecas usadas**:

1. **json**: Permite trabajar con datos en formato JSON (JavaScript Object Notation), útil para leer y escribir datos estructurados.
2. **from datetime import date, timedelta, datetime**: Importa clases de la biblioteca datetime para manejar fechas y horas.
   * date: Para trabajar con fechas sin tener en cuenta la hora.
   * timedelta: Para realizar operaciones de suma o resta entre fechas, o manipular intervalos de tiempo.
   * datetime: Para trabajar con fechas y horas completas (fecha y tiempo).
3. **import re**: Importa la biblioteca de expresiones regulares, que se utiliza para realizar búsquedas y manipulaciones complejas en cadenas de texto.
4. **from functools import reduce**: Importa la función reduce, que permite aplicar una función de manera acumulativa a los elementos de un iterable (como una lista), reduciéndolos a un solo valor.

**Funciones:**

**menu\_principal():**

1. **Muestra un banner de bienvenida**:  
   Utiliza un bloque de texto almacenado en la variable banner para imprimir un mensaje decorativo en la consola, dando la bienvenida al usuario.
2. **Despliega el menú**:  
   Muestra un menú con cinco opciones numeradas: consultar información de productos, gestionar productos (ABM), caja, estadísticas y salir.
3. **Solicita una opción al usuario**:  
   Pide al usuario que ingrese un número correspondiente a una de las opciones del menú.
4. **Validación de la entrada**:

* Si el número ingresado está entre 1 y 5, la función lo retorna como valor de salida.
* Si el número no es válido o si el usuario ingresa algo que no es un número entero, muestra un mensaje de error y vuelve a solicitar la entrada.

1. **Retorno**:  
   Devuelve el número de la opción seleccionada para que sea utilizado en el programa principal.

**leer\_archivo\_productos():**

1. **Apertura del archivo**:  
   Intenta abrir el archivo llamado productos.json en modo lectura ("r").
2. **Lectura del contenido**:  
   Si la apertura es exitosa, lee todo el contenido del archivo como una cadena de texto y lo almacena en la variable lineas\_productos. Luego, cierra el archivo para liberar el recurso.
3. **Verificación de datos**:
   * Si el archivo está vacío (lineas\_productos tiene longitud 0), muestra un mensaje indicando que no hay datos y termina la ejecución de la función sin devolver nada.
4. **Conversión del contenido**:  
   Convierte la cadena de texto leída del archivo en un objeto Python (generalmente un diccionario o lista) usando json.loads y lo guarda en la variable productos.
5. **Retorno**:  
   Devuelve el objeto productos para que se utilice en el programa principal.
6. **Manejo de errores**:  
   Si ocurre algún error durante la apertura o lectura del archivo, muestra un mensaje indicando que no se pudo abrir el archivo productos.json.

**leer\_archivo\_ventas():**

1. **Apertura del archivo**:  
   Intenta abrir el archivo llamado ventas.json en modo lectura ("r").
2. **Lectura del contenido**:  
   Si la apertura es exitosa, lee todo el contenido del archivo como una cadena de texto y lo almacena en la variable lineas\_ventas. Luego, cierra el archivo para liberar el recurso.
3. **Verificación de datos**:
   * Si el archivo está vacío (lineas\_ventas tiene longitud 0), muestra un mensaje indicando que no hay datos y termina la ejecución de la función sin devolver nada.
4. **Conversión del contenido**:  
   Convierte la cadena de texto leída del archivo en un objeto Python (generalmente un diccionario o lista) usando json.loads y lo guarda en la variable ventas.
5. **Retorno**:  
   Devuelve el objeto ventas para que se utilice en el programa principal.
6. **Manejo de errores**:  
   Si ocurre algún error durante la apertura o lectura del archivo, muestra un mensaje indicando que no se pudo abrir el archivo ventas.json.

**leer\_archivo\_promos():**

1. **Apertura del archivo**:  
   Intenta abrir el archivo llamado promos.json en modo lectura ("r").
2. **Lectura del contenido**:  
   Si la apertura es exitosa, lee todo el contenido del archivo como una cadena de texto y lo almacena en la variable lineas\_promos. Luego, cierra el archivo para liberar el recurso.
3. **Verificación de datos**:
   * Si el archivo está vacío (lineas\_promos tiene longitud 0), muestra un mensaje indicando que no hay datos y termina la ejecución de la función sin devolver nada.
4. **Conversión del contenido**:  
   Convierte la cadena de texto leída del archivo en un objeto Python (generalmente un diccionario o lista) usando json.loads y lo guarda en la variable promos.
5. **Retorno**:  
   Devuelve el objeto promos para que se utilice en el programa principal.
6. **Manejo de errores**:  
   Si ocurre algún error durante la apertura o lectura del archivo, muestra un mensaje indicando que no se pudo abrir el archivo promos.json.

**limpiar\_espacios(nombre\_producto):**

1. **Propósito**:  
   Elimina los espacios en blanco al inicio y al final de un texto dado.
2. **Entrada**:  
   Recibe como argumento una cadena de texto (nombre\_producto).
3. **Proceso**:  
   Usa la función re.sub de la librería re (expresiones regulares) para buscar y eliminar:
   * Los espacios en blanco al inicio (^\s+).
   * Los espacios en blanco al final (\s+$).
4. **Salida**:  
   Devuelve el texto sin espacios al inicio ni al final.

**menu\_info\_productos():**

1. **Llamada a leer\_archivo\_productos()**:  
   La función comienza llamando a leer\_archivo\_productos() para cargar los datos de productos desde el archivo productos.json. Los datos se almacenan en la variable productos.
2. **Obtención de las columnas**:  
   Se obtiene una lista de las claves (nombres de las columnas) del primer elemento en la lista de productos usando productos[0].keys(). Esto permite saber qué datos se van a mostrar en la tabla.
3. **Cálculo de anchos de columnas**:  
   Se crea un diccionario anchuras que almacena el ancho máximo de cada columna. El ancho se calcula comparando la longitud del nombre de cada columna con la longitud de los valores de cada fila en esa columna.  
   Se recorre cada fila de productos, actualizando el ancho de cada columna según sea necesario.
4. **Impresión de la cabecera**:  
   Se imprime la cabecera de la tabla, alineando los nombres de las columnas según su ancho calculado.  
   Se genera y muestra un separador entre la cabecera y las filas de datos.
5. **Impresión de los datos**:  
   Se recorren las filas de productos e imprime cada fila de datos, alineando los valores según el ancho de sus respectivas columnas.
6. **Menú de búsqueda**:  
   Se presenta al usuario un menú con opciones para realizar búsquedas:

1- Búsqueda por Nombre

2- Búsqueda por Marca

3- Búsqueda por Promoción

4- Volver  
Se solicita al usuario que ingrese una opción, validando que esté entre 1 y 4.

1. **Manejo de búsquedas**:  
   Si el usuario elige una opción de búsqueda (1, 2 o 3), se pide el valor a buscar y se llama a la función busqueda\_filtrada, pasando la columna correspondiente y el valor ingresado.  
   Después de cada búsqueda, se vuelve a mostrar el menú de búsqueda.
2. **Finalización**:  
   Si el usuario selecciona la opción 4 para volver, la función termina.

**busqueda\_filtrada(productos, columna\_busqueda, valor\_busqueda):**

1. **Descripción general**:  
   Esta función filtra los productos según la columna y el valor que elijas (nombre, marca o promoción). Si no encuentra resultados, muestra un mensaje indicándolo. Si encuentra coincidencias, muestra los resultados en una tabla.
2. **Obtención de las columnas**:  
   Se obtiene una lista de las claves (nombres de las columnas) del primer elemento de la lista de productos, utilizando productos[0].keys(). Esto nos permite saber qué columnas tiene el archivo de productos, ya que es una lista de diccionarios.
3. **Filtrado de los datos**:  
   Se crea una lista datos\_filtrados mediante una list comprehension, en la cual se recorren todos los productos y se verifica si el valor\_busqueda (en minúsculas) aparece en la columna seleccionada (también en minúsculas). Para cada producto, se utiliza fila.get(columna\_busqueda, '') para obtener el valor de la columna deseada (si no existe, se pasa un valor vacío) y se compara con el valor de búsqueda.
4. **Verificación de resultados**:  
   Si datos\_filtrados está vacío (es decir, no se encontraron productos que coincidan con la búsqueda), se imprime un mensaje diciendo que no se encontraron resultados para el valor de búsqueda en la columna seleccionada. Luego, la función termina.
5. **Cálculo del ancho de las columnas**:  
   Se crea un diccionario anchuras donde se guarda el ancho de cada columna. El valor inicial es la longitud del nombre de la columna. Luego, se recorren las filas de datos\_filtrados y se actualiza el ancho de cada columna para que sea suficiente para mostrar tanto el nombre de la columna como el valor más largo de esa columna en los productos filtrados.
6. **Impresión de la cabecera y separador**:
   * Cabecera: Se construye la cabecera de la tabla concatenando los nombres de las columnas con el separador " | ". Para asegurar que cada columna esté alineada correctamente, se utiliza ljust() para alinear los nombres de las columnas a la izquierda, utilizando el ancho calculado previamente.
   * Separador: Se crea una línea de separación entre la cabecera y los datos. Se genera una cadena de guiones cuyo largo es igual al ancho de cada columna, separada por "-+-".
7. **Impresión de los datos filtrados**:  
   Finalmente, se recorre la lista datos\_filtrados y, para cada fila, se construye una cadena que contiene los valores de las columnas separados por " | ". Al igual que con la cabecera, los valores se alinean a la izquierda utilizando ljust() con el ancho correspondiente. Se imprime cada fila de productos.

**menu\_abm\_productos():**

1. **Descripción general**:  
   Esta función muestra un menú interactivo que permite al usuario realizar acciones sobre los productos, como agregar, eliminar o modificar. El usuario elige una opción y la función ejecuta la acción correspondiente.
2. **Lectura de productos**:  
   Se llama a la función leer\_archivo\_productos() para cargar la lista de productos desde el archivo o base de datos. Los productos se almacenan en la variable productos.
3. **Bucle principal (while True)**:  
   La función entra en un bucle infinito, que permite al usuario elegir acciones repetidamente hasta que se seleccione la opción de salir (volver). Esto es útil para mantener el menú activo mientras el usuario no decida salir.
4. **Mostrar el menú**:  
   Dentro del bucle, se muestra el menú de opciones:
   * 1. Alta Producto: Para agregar un nuevo producto.
   * 2. Baja Producto: Para eliminar un producto.
   * 3. Modificación Producto: Para modificar un producto existente.
   * 4. Volver: Para salir del menú y volver al nivel anterior.
5. **Ingreso de opción**:  
   Se solicita al usuario que ingrese un número correspondiente a una de las opciones del menú. Esta entrada se convierte en un número entero con int(input()).
6. **Validación de opción**:  
   Si el usuario ingresa un valor fuera del rango (no entre 1 y 4), se muestra un mensaje de error y se solicita que ingrese una opción correcta.
7. **Ejecutar acción según la opción seleccionada**:
   * Si el usuario selecciona 1, se llama a la función alta\_producto(productos) para agregar un nuevo producto.
   * Si el usuario selecciona 2, se llama a la función baja\_producto(productos) para eliminar un producto.
   * Si el usuario selecciona 3, se llama a la función modificar\_producto(productos) para modificar un producto existente.
   * Si el usuario selecciona 4, se usa return para salir de la función, lo que termina el bucle y vuelve al nivel anterior del programa.
8. **Manejo de errores (excepción ValueError)**:  
   Si el usuario ingresa un valor que no es un número entero (por ejemplo, una letra o un símbolo), se captura la excepción ValueError y se muestra el mensaje "Error. Debe ingresar un número entero." Esto asegura que solo se procesen entradas válidas.

**busqueda\_producto\_alta(productos):**

1. **Descripción general**:  
   Esta función permite al usuario ingresar el nombre y la marca de un producto para darlo de alta. Antes de proceder con el alta, verifica que el producto no exista ya en la lista de productos. Si el producto ya existe, solicita nuevos datos hasta que se ingrese un producto único.
2. **Inicialización de la variable encontrado**:  
   Se inicializa la variable encontrado como False, indicando que aún no se ha encontrado un producto con el nombre y la marca ingresados.
3. **Ingreso del nombre del producto**:
   * Se solicita al usuario que ingrese el nombre del producto.
   * El nombre ingresado se limpia de espacios alrededor con la función limpiar\_espacios().
   * Si el nombre no está vacío (es decir, el usuario no ingresó solo espacios), se guarda el valor limpio y se sale del bucle.
   * Si el nombre está vacío, se muestra un mensaje de error y se vuelve a solicitar el nombre.
4. **Ingreso de la marca del producto**:
   * De forma similar al nombre, se solicita la marca del producto.
   * Se limpia el valor de los espacios con limpiar\_espacios() y se valida que no esté vacío.
   * Si la marca está vacía, se muestra un mensaje de error y se vuelve a pedir la marca.
5. **Verificación de existencia del producto**:
   * Se recorre la lista de productos, verificando si ya existe un producto con el mismo nombre y marca (se compara ignorando mayúsculas y minúsculas).
   * Si se encuentra un producto con el mismo nombre y marca, la variable encontrado se cambia a True, indicando que el producto ya está registrado.
   * Si no se encuentra el producto, se procede a continuar con el alta.
6. **Control de duplicados**:
   * Si se encontró un producto con el mismo nombre y marca (es decir, encontrado es True), se vuelve a solicitar al usuario un nombre y una marca nuevos.
   * Se repite el proceso de validación del nombre y la marca, limpiando los espacios y asegurándose de que no estén vacíos.
   * Luego, se vuelve a verificar si el producto ingresado ya existe en la lista.
   * El proceso de búsqueda y validación se repite hasta que se ingrese un nombre y una marca únicos (es decir, hasta que no se encuentre el producto).
7. **Retorno del nombre y la marca**:  
   Finalmente, cuando el producto es único y no se encuentra en la lista, se retorna el nombre y marca del producto para ser utilizado en el alta del mismo.

**busqueda\_producto\_baja(productos):**

**Solicita el ID del producto:** La función comienza pidiendo al usuario que ingrese el **ID** del producto que desea eliminar. Si el usuario ingresa algo que no es un número, muestra un mensaje de error y vuelve a pedir el ID.

**Verifica si el producto existe:** Después de recibir el ID, recorre la lista de productos para verificar si el ID ingresado corresponde a algún producto. Si lo encuentra, establece la variable encontrado en **True**.

**Manejo de errores:** Si no encuentra el producto con el ID ingresado, muestra un mensaje de error y sigue pidiendo al usuario un ID válido hasta que lo encuentre.

**Retorno:** Una vez que se encuentra el producto o se valida que existe, la función retorna el **ID** del producto y un valor **True** o **False**, dependiendo de si se encontró el producto.

**busqueda\_producto\_modificacion(productos):**

1. **Solicitar ID**: Pide al usuario que ingrese el ID del producto que desea modificar. Si no es un número válido, vuelve a pedirlo.
2. **Buscar el producto**: Recorre la lista de productos para buscar el producto con el ID ingresado. Si lo encuentra, lo guarda en la variable encontrado.
3. **Manejo de error**: Si no se encuentra el producto con ese ID, muestra un mensaje de error y sigue pidiendo el ID hasta que se encuentre un producto válido.
4. **Retornar el producto**: Una vez encontrado el producto, retorna el diccionario del producto encontrado (encontrado).

**busqueda\_producto\_caja(productos):**

1. **Solicitar ID**: Pide al usuario que ingrese el ID del producto que quiere vender. Si el valor ingresado no es un número válido, vuelve a pedirlo.
2. **Buscar el producto**: Recorre la lista de productos y busca el producto con el ID ingresado. Si lo encuentra, lo guarda en la variable encontrado.
3. **Manejo de error**: Si no se encuentra el producto con ese ID, muestra un mensaje de error y sigue pidiendo el ID hasta que se encuentre un producto válido.
4. **Retornar el producto**: Una vez encontrado el producto, retorna el diccionario del producto encontrado (encontrado).

**alta\_producto(productos):**

1. **Obtener ID del nuevo producto**: Se busca el producto con el ID más alto de la lista de productos. Luego, el ID para el nuevo producto se establece como uno mayor al ID máximo encontrado.
2. **Solicitar datos del producto**:
   * Nombre y Marca: Se obtienen usando la función busqueda\_producto\_alta().
   * Precio: Se pide al usuario que ingrese el precio del producto, validando que sea un número positivo.
   * Ubicación: Se solicita la ubicación del producto, asegurándose de que no esté vacía ni contenga solo espacios.
   * Stock: Se pide el stock del producto, asegurando que sea un número entero mayor o igual a 1.
3. **Opcional**: Agregar una promoción: Se pregunta al usuario si quiere agregar una promoción al producto. Si la respuesta es "Sí", se le presenta un menú de opciones de promoción y el usuario debe ingresar los detalles de la promoción. Si elige "No", la promoción se establece como "0" (sin promoción).
4. **Agregar el producto**: Se guarda el nuevo producto como un diccionario con todos los datos proporcionados (ID, nombre, marca, precio, ubicación, stock, promoción) en la lista de productos.
5. **Guardar en archivos JSON**:
   * Productos: La lista de productos se convierte a formato JSON y se guarda en el archivo productos.json.
   * Promociones: Las promociones se convierten a formato JSON y se guardan en el archivo promos.json.
6. **Manejo de errores**: Se utiliza un bloque try-except para manejar cualquier error al guardar los archivos JSON. Si algo sale mal, se imprime un mensaje de error.

**obtener\_id(elemento):**

1. **Propósito de la función**: Esta función toma un diccionario de producto como argumento y devuelve el valor de su campo id.
2. **¿Por qué se utiliza?** Se usa principalmente como una función de apoyo para encontrar el ID más alto cuando estamos agregando un nuevo producto. En otras palabras, se utiliza en la función alta\_producto para identificar el producto con el ID máximo, a fin de asignar un nuevo ID único al nuevo producto.
3. **¿Qué hace exactamente?**
   * Toma un diccionario (por ejemplo, un producto) como argumento.
   * Extrae el valor asociado a la clave id de ese diccionario.
   * Devuelve el valor de la clave id.

**baja\_producto(productos):**

1. **Mostrar los productos disponibles**:
   * Llama a la función mostrar\_info\_productos() para mostrar la lista de productos que están disponibles, permitiendo al usuario ver cuáles están en el sistema y decidir cuál desea eliminar.
2. **Buscar el producto a eliminar**:
   * Se llama a la función busqueda\_producto\_baja(productos), que devuelve el ID del producto a eliminar y un booleano (encontrado) que indica si el producto con ese ID fue encontrado en la lista.
3. **Eliminar el producto**:
   * Si el producto fue encontrado (encontrado es verdadero), se utiliza una comprensión de lista para crear una nueva lista de productos, excluyendo el producto con el ID dado (producto['id'] != idProducto). Esto elimina efectivamente el producto de la lista.
4. **Guardar los cambios en el archivo JSON**:
   * La lista actualizada de productos (productos\_actualizados) se convierte en una cadena JSON utilizando json.dumps(), con una indentación de 4 espacios para mejorar su legibilidad.
5. **Manejo de excepciones**:
   * Se abre el archivo productos.json en modo escritura, y se guarda la lista actualizada de productos en ese archivo. Si ocurre algún error durante el proceso (por ejemplo, si no se puede escribir en el archivo), se captura la excepción y se muestra un mensaje de error.
6. **Confirmación o error**:
   * Si los cambios se guardan correctamente en el archivo, se muestra el mensaje "Producto eliminado con éxito". Si hay un error, se muestra el mensaje "No se pudo eliminar el producto en el archivo productos".

**modificar\_producto(productos):**

1. **Mostrar los productos disponibles**:
   * Llama a la función mostrar\_info\_productos() para mostrar la lista de productos existentes, ayudando al usuario a decidir cuál producto desea modificar.
2. **Buscar el producto a modificar**:
   * Se llama a la función busqueda\_producto\_modificacion(productos) para encontrar el producto que se desea modificar. La función devuelve un diccionario con la información del producto si es encontrado.
3. **Menú de opciones de modificación**:
   * Se presenta un menú para que el usuario elija qué campo del producto desea modificar. Las opciones son:
     + 1: Nombre
     + 2: Marca
     + 3: Precio
     + 4: Ubicación
     + 5: Stock
     + 6: Promoción
     + 7: Finalizar (guardar y salir)
4. **Validar opción seleccionada**:
   * El programa valida que la opción ingresada esté entre 1 y 7. Si no, pide que ingrese una opción válida.
5. **Modificar los campos del producto**:
   * Para cada opción (1 a 6), el programa solicita el nuevo valor del campo a modificar:
     + Nombre y marca: El valor ingresado se limpia de espacios en blanco con la función limpiar\_espacios() y se valida que no esté vacío.
     + Precio y stock: Se valida que el valor sea numérico y que el precio sea mayor a 0 y el stock mayor o igual a 1.
     + Promoción: Se muestra un submenú para agregar una promoción, permitiendo al usuario seleccionar entre varias opciones de promoción, y luego se valida que la opción ingresada sea válida. La promoción se almacena en un conjunto de promociones para evitar duplicados.
6. **Finalizar y guardar los cambios**:
   * Si el usuario elige finalizar (opción 7), el programa guarda todos los cambios realizados en el archivo productos.json y en el archivo promos.json.
     + La lista de productos se convierte a formato JSON utilizando json.dumps(), y luego se guarda en productos.json.
     + Las promociones se convierten también en formato JSON y se guardan en promos.json.
   * Si el archivo no se puede guardar debido a algún error, se captura la excepción y se muestra un mensaje de error.
7. **Manejo de excepciones**:
   * Se utiliza un bloque try-except para manejar posibles errores durante la escritura en los archivos, como problemas de permisos o archivos bloqueados.

**mostrar\_info\_productos():**

**Leer productos**: Obtiene la lista de productos desde un archivo con leer\_archivo\_productos().

**Obtener columnas**: Extrae los nombres de las columnas del primer producto.

**Calcular anchos**: Calcula el ancho máximo de cada columna, considerando tanto los nombres como los valores.

**Imprimir cabecera**: Genera y muestra la cabecera con los nombres de las columnas, alineados.

**Imprimir separador**: Muestra una línea de separación que coincide con el ancho de cada columna.

**Imprimir filas**: Muestra los datos de cada producto alineados bajo las columnas correspondientes.

**menu\_caja():**

1. **Preparar el entorno inicial**:
   * Leer el archivo ventas.json para cargar las ventas existentes.
   * Obtener y formatear la fecha actual, que se usará para las ventas.
2. **Mostrar el menú principal**:
   * Presentar opciones al usuario:
     1. Ingresar productos a la venta.
     2. Volver al menú principal.
3. **Procesar la venta de productos**:
   * Leer el archivo productos.json para cargar el inventario.
   * Mostrar información detallada de los productos.
   * Inicializar una lista para registrar los productos vendidos y un importe total de la venta.
4. **Registrar productos vendidos**:
   * Buscar un producto por su ID utilizando la función busqueda\_producto\_caja().
   * Solicitar al usuario la cantidad a comprar y verificar si hay suficiente stock.
   * Calcular el importe del producto considerando posibles promociones mediante obtener\_importe().
5. **Actualizar las ventas y el stock**:
   * Crear un diccionario con los detalles de la venta y agregarlo a la lista de ventas.
   * Escribir los datos actualizados en ventas.json.
   * Restar la cantidad vendida del stock del producto y actualizar productos.json.
6. **Mostrar el resumen de la venta**:
   * Crear una tabla que incluya detalles de los productos vendidos.
   * Calcular el total de la venta y mostrarlo de forma formateada.
7. **Seleccionar método de pago**:
   * Permitir al cliente elegir entre efectivo o tarjeta.
   * Si se selecciona efectivo, calcular el vuelto si el cliente paga más del importe total.
8. **Permitir agregar más productos o finalizar**:
   * Preguntar al usuario si desea añadir más productos.
   * Si no, finalizar la venta mostrando el resumen completo.
9. **Opciones de salida**:
   * Si el usuario elige volver al menú principal, terminar el bucle.

**def obtener\_importe(importe, promocion, cantidad):** La función calcula el importe total de un producto considerando la cantidad adquirida y el tipo de promoción que aplique.

1. **Reconocimiento de promociones tipo "MxN" (ejemplo: "2x3"):**
   * Identifica ofertas donde al comprar una cierta cantidad de productos (por ejemplo, 3), solo se paga por una cantidad menor (por ejemplo, 2).
   * La función calcula cuántos grupos de la promoción se pueden aplicar según la cantidad comprada. Luego suma el costo de estos grupos y el costo de las unidades que no entran en la promoción.
2. **Reconocimiento de promociones tipo "descuento en una unidad específica" (ejemplo: "32% de descuento en la 5ta unidad"):**
   * Identifica promociones donde un porcentaje de descuento aplica en una unidad específica (como la quinta).
   * La función suma el precio completo de las demás unidades y aplica el descuento solo a la unidad indicada.
3. **Reconocimiento de promociones tipo "descuento general" (ejemplo: "35% en todas las unidades"):**
   * Identifica descuentos que aplican a todas las unidades compradas.
   * La función multiplica el precio total de los productos por el porcentaje restante después del descuento.
4. **Sin promoción:**
   * Si no hay promoción válida, calcula simplemente el precio total multiplicando la cantidad por el precio por unidad.

**formato\_fecha(fecha):**

**1) Lista de meses del año**  
Se crea una tupla llamada meses que contiene los nombres de los 12 meses en orden, desde "Enero" hasta "Diciembre".  
Esto será usado para convertir el número del mes (como 1, 2, 3, etc.) en el nombre correspondiente.

**2) Extraer el día de la fecha**  
La función utiliza fecha.day para obtener el día del mes de la fecha proporcionada. Por ejemplo, si la fecha es 2024-09-23, el día sería 23.

**3) Obtener el mes en texto**

* fecha.month da el número del mes (por ejemplo, 9 para septiembre).
* Se resta 1 a ese número porque las tuplas comienzan en el índice 0.
* Luego, se busca el nombre del mes en la tupla meses. Por ejemplo, si el mes es 9, el resultado será "Septiembre".

**4) Extraer el año de la fecha**  
Con fecha.year, se obtiene el año completo de la fecha. Por ejemplo, de 2024-09-23, se obtendrá el año 2024.

**5) Formatear el mensaje**  
Se combinan el día, el nombre del mes y el año en una cadena de texto que sigue el formato:  
"día de mes del año".  
Por ejemplo, para 2024-09-23, el resultado será: "23 de Septiembre del 2024".

**6) Devolver el resultado**  
Finalmente, la función devuelve el mensaje formateado como resultado.

**menu\_estadisticas():**

**1. Preparación inicial**

* Fecha actual: Se obtiene la fecha del día con date.today() y se guarda como cadena (str), para usarla en filtrados y en el menú.
* Ventas registradas: Se utiliza leer\_archivo\_ventas() para cargar las ventas almacenadas en un archivo, devolviendo una lista de diccionarios que contienen información de las ventas.

**2. Mostrar el menú principal**

* Se presenta un menú principal con opciones para generar estadísticas (diarias, mensuales, anuales, etc.).
* Validación de entrada: Si el usuario introduce una opción inválida, se le pide que reingrese un valor correcto.

**3. Opciones del menú principal**

Dependiendo de la opción seleccionada, la función realiza una tarea específica:

Opción 1: Estadísticas diarias (hoy)

1. Filtra las ventas del día actual usando filter con una función lambda que compara la fecha de la venta con la fecha actual.
2. Si no hay ventas registradas, muestra un mensaje y vuelve al menú principal.
3. Si hay ventas, presenta un submenú con opciones para:
   * Mostrar la cantidad total de ventas.
   * Mostrar las 3 promociones más utilizadas.
   * Calcular la recaudación total.
   * Mostrar la cantidad total de productos vendidos.
   * Mostrar las 3 marcas más vendidas.
   * Volver al menú principal.

Opción 2: Estadísticas diarias con fecha específica

1. Se obtiene una fecha personalizada ingresada por el usuario (usando una función externa obtener\_fecha()).
2. Filtra las ventas que coinciden con esa fecha.
3. Si no hay ventas registradas, vuelve al menú principal.
4. Si hay ventas, presenta un submenú similar al de la opción 1.

Opción 3: Estadísticas mensuales (mes pasado)

1. Calcula el mes pasado y el año actual.
2. Usa expresiones regulares (re.compile) para filtrar las ventas que coinciden con el mes anterior y el año actual.
3. Si no hay ventas, muestra un mensaje y vuelve al menú principal.
4. Si hay ventas, presenta un submenú similar al de las opciones anteriores.

Opción 4: Estadísticas mensuales con mes a elección

1. Solicita al usuario que ingrese un mes específico.
2. Usa expresiones regulares para filtrar las ventas que coinciden con ese mes y el año actual.
3. Si no hay ventas, vuelve al menú principal.
4. Si hay ventas, muestra un submenú similar al de las opciones anteriores.

Opción 5: Estadísticas anuales con año a elección

1. Solicita al usuario que ingrese un año específico.
2. Usa expresiones regulares para filtrar las ventas que coinciden con el año ingresado.
3. Si no hay ventas, vuelve al menú principal.
4. Si hay ventas, muestra un submenú similar al de las opciones anteriores.

Opción 6: Salir

* Termina la función y regresa al punto de llamada.

**Validaciones generales**

* Las opciones del menú se validan para asegurarse de que estén dentro del rango permitido (1 a 6).
* Se capturan excepciones (ValueError) para manejar entradas no numéricas.

**Funciones auxiliares**

La función utiliza funciones externas como:

* leer\_archivo\_ventas(): Lee el archivo de ventas.
* mostrar\_top\_promociones(): Muestra las 3 promociones más utilizadas.
* mostrar\_recaudacion\_total\_dia\_mes(): Calcula la recaudación total.
* mostrar\_total\_productos\_vendidos(): Muestra el número total de productos vendidos.
* mostrar\_top\_marcas(): Muestra las 3 marcas más vendidas.

**obtener\_fecha():** Solicita al usuario que ingrese una fecha en el formato YYYY-MM-DD y valida que:

1. El formato sea correcto.
2. La fecha no sea mayor a la fecha actual.

Si ambas condiciones se cumplen, devuelve la fecha ingresada como un objeto date. En caso de error, solicita al usuario que intente nuevamente.

**Inicio de un bucle infinito (while True):**

* Permite repetir la solicitud al usuario hasta que se ingrese una fecha válida.

**Solicitar entrada al usuario**:

* fecha\_str = input("Ingresa una fecha (formato: YYYY-MM-DD): ")
* Se pide al usuario que ingrese una fecha en formato de texto.

**Intentar convertir la entrada en un objeto de fecha**:

* fecha\_ingresada = datetime.strptime(fecha\_str, "%Y-%m-%d").date()
* Se intenta interpretar la cadena fecha\_str como una fecha en el formato YYYY-MM-DD.
* Si el formato no coincide, salta al bloque except ValueError.

**Verificar que la fecha no sea futura**:

* if fecha\_ingresada > date.today():
  + Comprueba si la fecha ingresada es mayor que la fecha actual.
  + Si es futura, se muestra un mensaje de error y el bucle continúa.

**Devolver la fecha válida**:

* Si el formato es correcto y la fecha no es futura, la función retorna la fecha como un objeto date.

**Manejar errores de formato**:

* Si la conversión falla (por ejemplo, el usuario escribe "2024/11/19" en lugar de "2024-11-19"), se muestra un mensaje:
  + "Fecha inválida. Por favor ingresa una fecha en el formato correcto (YYYY-MM-DD)."
* El bucle repite la solicitud.

**mostrar\_recaudacion\_total\_dia\_mes():** Recibe una lista de ventas filtradas y calcula la recaudación total del mes, así como el total recaudado por día. Luego, imprime estos resultados.

**1) Inicio de la estructura try-except:** Se utiliza un bloque try-except para manejar posibles errores durante la ejecución, como problemas con los datos de las fechas. Si ocurre un error, el bloque except captura el error y muestra un mensaje de error.

**2) Extracción de los importes:** Utiliza la función map y una función lambda para crear una nueva lista llamada recaudaciones. Esta lista contiene los valores de "importe" de cada venta filtrada. Se utiliza lambda para obtener solo el valor de "importe" de cada venta en la lista ventas\_filtradas.

**3) Cálculo de la recaudación total:** La función sum() toma la lista recaudaciones y calcula la suma de todos los importes, es decir, la recaudación total del mes.

**4) Agrupación de importes por día:** Se crea un diccionario importes\_por\_dia donde las claves son los días (extraídos de la fecha de cada venta) y los valores son los importes correspondientes. El ciclo for recorre cada venta en ventas\_filtradas, extrae el día de la fecha (los dos últimos caracteres) y agrega el importe correspondiente al día. Si el día ya existe en el diccionario, se suma el importe; si no, se crea una nueva entrada con el importe de esa venta.

**5) Conversión del diccionario a una lista:** Después de agrupar los importes por día, la función crea una lista de listas llamada matriz\_dias\_importes. Cada sublista contiene dos elementos: el día y el importe total de ventas para ese día. Se utiliza sorted() para ordenar esta lista de acuerdo con los días.

**6) Impresión de los resultados:** Primero, se imprime el encabezado "Recaudación" y "Día, Importe". Luego, se recorre la lista matriz\_dias\_importes e imprime cada fila, que representa un día y su recaudación total. Finalmente, se imprime la recaudación total del mes.

**7) Manejo de errores:** Si ocurre un error al intentar extraer los importes o calcular la recaudación (por ejemplo, si hay un valor incorrecto en las fechas), el bloque except captura el error y muestra el mensaje "No se pudo obtener la recaudación total."

**mostrar\_recaudacion\_total\_año():** Recibe una lista de ventas filtradas y calcula la recaudación total del año, así como el total recaudado por mes. Luego, imprime estos resultados.

1. **Inicio de la estructura try-except:** Al igual que la función anterior, se utiliza un bloque try-except para manejar posibles errores durante la ejecución, como problemas con los datos de las fechas. Si se produce un error, el bloque except captura el error y muestra un mensaje de error.
2. **Extracción de los importes**: Se utiliza map con una función lambda para crear una lista llamada recaudaciones, que contiene los valores de "importe" de cada venta en ventas\_filtradas. Este paso asegura que solo se extraigan los importes de las ventas.
3. **Cálculo de la recaudación total**: Se usa la función sum() para calcular la suma total de los importes, es decir, la recaudación total del año.
4. **Agrupación de importes por mes**: Se crea un diccionario importes\_por\_mes, donde las claves son los meses (extraídos de la fecha de cada venta) y los valores son los importes correspondientes a ese mes. El ciclo for recorre cada venta, extrae el mes de la fecha (caracteres de la posición 5 a 7) y agrega el importe de la venta al mes correspondiente. Si el mes ya existe en el diccionario, se suma el importe; si no, se crea una nueva entrada con el importe de esa venta.
5. C**onversión del diccionario a una lista**: Después de agrupar los importes por mes, la función convierte el diccionario en una lista de listas llamada matriz\_meses\_importes. Cada sublista contiene dos elementos: el mes y el importe total de ventas para ese mes. La lista se ordena por mes utilizando sorted().
6. **Impresión de los resultados**: Primero, se imprime el encabezado "Recaudación" y "Mes, Importe". Luego, se recorre la lista matriz\_meses\_importes e imprime cada fila, que representa un mes y su recaudación total. Finalmente, se imprime la recaudación total del año.
7. **Manejo de errores**: Si ocurre un error al intentar extraer los importes o calcular la recaudación (por ejemplo, si hay un valor incorrecto en las fechas), el bloque except captura el error y muestra el mensaje "No se pudo obtener la recaudación total."

**mostrar\_total\_productos\_vendidos():** Recibe una lista de ventas filtradas y calcula la cantidad total de productos vendidos por cada producto. Luego, imprime estos resultados.

1. **Inicio de la estructura try-except**: Se utiliza un bloque try-except para manejar posibles errores durante la ejecución, como problemas con los datos de las fechas o la cantidad de ventas. Si ocurre un error, el bloque except captura el error y muestra un mensaje de error.
2. **Agrupación de cantidades por producto**: Se crea un diccionario cantidades\_por\_producto, donde las claves son los nombres de los productos (extraídos de cada venta) y los valores son las cantidades vendidas de esos productos. El ciclo for recorre cada venta en ventas\_filtradas, extrae el nombre del producto y suma la cantidad vendida de ese producto en el diccionario. Si el producto ya existe en el diccionario, se suma la cantidad; si no, se crea una nueva entrada con la cantidad de esa venta.
3. **Conversión del diccionario a una lista**: Después de agrupar las cantidades por producto, la función convierte el diccionario en una lista de listas llamada matriz\_productos\_cantidades. Cada sublista contiene dos elementos: el nombre del producto y la cantidad total vendida. Esta lista se construye con una comprensión de listas.
4. **Impresión de los resultados**: Primero, se imprime el encabezado "Producto, Cantidad". Luego, se recorre la lista matriz\_productos\_cantidades e imprime cada fila, que representa un producto y su cantidad total vendida.
5. **Manejo de errores**: Si ocurre un error al intentar agrupar las cantidades o al procesar las ventas (por ejemplo, si hay un valor incorrecto en las cantidades), el bloque except captura el error y muestra el mensaje "No se pudo obtener el total de productos vendidos."

**mostrar\_top\_marcas():** Recibe una lista de ventas filtradas y calcula las 3 marcas más frecuentes (más vendidas) en las ventas. Luego, imprime el nombre de esas marcas.

**Inicio de la estructura try-except:** Se utiliza un bloque try-except para manejar posibles errores durante la ejecución, como problemas con los datos de las ventas. Si ocurre un error, el bloque except captura el error y muestra un mensaje indicando que no se pudo obtener el top 3 de marcas.

**Extracción de las marcas:** Utiliza la función map junto con una función lambda para extraer solo los valores de la clave "marca" de cada venta. map aplica la función lambda venta: venta["marca"] a cada venta en la lista ventas\_filtradas, devolviendo una lista con las marcas de todos los productos vendidos. Este resultado se convierte en una lista con list(...).

**Cálculo de las marcas más frecuentes:** La función crea una lista de tuplas con cada marca y su frecuencia de aparición. Para hacer esto:

* Se recorre cada marca única en la lista marcas usando set(marcas) (lo que elimina los duplicados).
* Para cada marca, se cuenta cuántas veces aparece en la lista original marcas con marcas.count(marca).
* La salida es una lista de tuplas (marca, frecuencia), donde la frecuencia es cuántas veces aparece cada marca.

**Ordenamiento de las marcas por frecuencia:** La función sorted() ordena esta lista de tuplas en orden descendente según la frecuencia. El parámetro key=lambda x: x[1] indica que la lista debe ser ordenada por el segundo valor de cada tupla (que es la frecuencia). reverse=True asegura que el orden sea de mayor a menor frecuencia.

**Selección de las 3 marcas más frecuentes**: Después de ordenar la lista de tuplas, la función toma solo los primeros 3 elementos (las 3 marcas más frecuentes) utilizando el índice [:3].

**Impresión de los resultados**: Se imprime el encabezado "Top 3 Marcas" y luego se recorre la lista top\_3\_marcas con un ciclo for y enumerate(). Para cada marca y su frecuencia, se imprime su posición en el top 3 (con el índice i + 1 para numerar correctamente desde 1).

**Manejo de errores**: Si ocurre un error en cualquier parte del proceso (como un problema con la extracción de las marcas o la manipulación de los datos), el bloque except captura el error y muestra el mensaje "No se pudo obtener el top 3 de marcas."

**mostrar\_top\_promociones():** Recibe una lista de ventas filtradas y calcula las 3 promociones más frecuentes (más aplicadas) en esas ventas. Luego, imprime las promociones más populares.

**Inicio de la estructura try-except**: Se utiliza un bloque try-except para manejar posibles errores durante la ejecución. Si hay problemas, como datos mal formateados o algún otro error durante la manipulación de las ventas, el bloque except captura el error y muestra el mensaje "No se pudo obtener el top 3 de promociones."

**Extracción de las promociones**: Se utiliza la función map con una función lambda para extraer solo los valores de la clave "promocion" de cada venta en la lista ventas\_filtradas. Esto crea una lista llamada promociones que contiene todas las promociones aplicadas en las ventas.

**Generación de la lista de tuplas (promoción, frecuencia):** La función genera una lista de tuplas, donde cada tupla contiene el nombre de una promoción y su frecuencia (cuántas veces aparece). Para hacerlo:

* Se recorre cada promoción única en la lista promociones utilizando set(promociones), lo que elimina duplicados y obtiene solo las promociones únicas.
* Para cada promoción, se utiliza filter junto con una función lambda para contar cuántas veces aparece esa promoción en la lista original promociones.

**Ordenamiento de las promociones por frecuencia**: La lista de tuplas se ordena con sorted(), utilizando el parámetro key=lambda x: x[1] para indicar que el orden debe hacerse según la frecuencia (que está en la segunda posición de cada tupla). El parámetro reverse=True asegura que las promociones más frecuentes aparezcan al principio.

**Selección de las 3 promociones más frecuentes**: Después de ordenar las promociones, la función toma los primeros 3 elementos de la lista con [:3], seleccionando así las 3 promociones más frecuentes.

**Impresión de los resultados**: La función imprime el encabezado "Top 3 Promociones". Luego, recorre la lista top\_3\_promociones con un ciclo for y enumerate(), imprimiendo el número de cada promoción (con el índice i + 1) seguido del nombre de la promoción.

**Manejo de errores**: Si ocurre algún error en el proceso (por ejemplo, problemas con los datos de las ventas o las promociones), el bloque except captura el error y muestra el mensaje "No se pudo obtener el top 3 de promociones."

**menu\_salir():** Tiene como objetivo mostrar un banner con un mensaje de despedida al usuario utilizando un print. Este mensaje está diseñado para dar un toque visual al momento de salir o finalizar una aplicación.

**Contenido del banner**: Dentro de la función, se define una cadena de texto llamada banner. Esta cadena contiene un mensaje de despedida formateado en arte ASCII, que se ve como un texto estilizado. El texto en el banner dice "Hasta Luego!" y está formado por varios caracteres para crear un efecto visual atractivo.

**Mostrar el banner**: Después de definir el banner, la función utiliza print(banner) para mostrar el contenido de la variable banner en la salida estándar (en pantalla).

**main():** Es el punto de entrada principal del programa. Su objetivo es interactuar con el usuario y gestionar las opciones que este elige en el menú principal. Según la opción seleccionada, ejecuta diferentes funciones y, en algunos casos, vuelve a mostrar el menú principal para que el usuario elija una nueva opción.

1. **Llamado al menú principal**: Se llama a la función menu\_principal(), que probablemente presenta un menú al usuario con varias opciones (como consultar productos, realizar operaciones de ABM, administrar la caja, consultar estadísticas, o salir). La opción seleccionada por el usuario se guarda en la variable opcion\_menu\_principal.
2. **Condiciones para seleccionar opciones**: Dependiendo del valor de opcion\_menu\_principal, se ejecuta un bloque de código específico:
   * Opción 1 (Consultar productos): Si el usuario selecciona la opción 1, se llama a la función menu\_info\_productos(), que seguramente permite al usuario consultar información sobre productos. Luego, se vuelve a llamar a main() para mostrar nuevamente el menú principal.
   * Opción 2 (Realizar ABM productos): Si el usuario selecciona la opción 2, se llama a la función menu\_abm\_productos(), que probablemente permite al usuario realizar operaciones de alta, baja y modificación (ABM) sobre productos. Al finalizar, se vuelve a llamar a main() para mostrar nuevamente el menú principal.
   * Opción 3 (Administrar caja): Si el usuario selecciona la opción 3, se llama a la función menu\_caja(), que gestiona las operaciones de caja o ventas. Al finalizar, también se vuelve a llamar a main() para mostrar nuevamente el menú principal.
   * Opción 4 (Consultar estadísticas): Si el usuario selecciona la opción 4, se llama a la función menu\_estadisticas(), que seguramente muestra estadísticas sobre las ventas, productos, etc. Después de mostrar las estadísticas, se vuelve a llamar a main() para que el usuario pueda elegir otra opción.
3. **Opción "Salir"**: Si el usuario selecciona una opción diferente a las anteriores (lo que indica que quiere salir), se llama a la función menu\_salir(), que muestra un mensaje de despedida, y el programa termina.
4. **Recursividad**: La función main() se llama a sí misma después de cada acción para permitir al usuario elegir otra opción del menú principal. Esto crea un ciclo en el que el programa sigue pidiendo opciones hasta que el usuario decide salir.
5. **Flujo de ejecución**: El flujo sigue este patrón:
   * El usuario selecciona una opción en el menú.
   * El programa ejecuta la acción correspondiente.
   * Después de completar la acción, vuelve a mostrar el menú principal llamando nuevamente a main().