ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

ESTUDIANTES

Mateo Rodrigez

Dennes Molina

Erik Villagomez

Ivonne Chauca

Diego Toapanta

David Guato

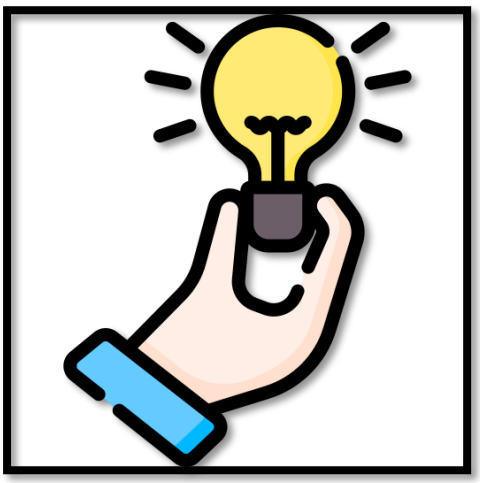
ALGORITMOS Y ESTRUCTURAS DE DATOS

2026/02/04

2. Introducción y Problemática

Problemática:
Las pequeñas tiendas y negocios locales enfrentan dificultades para gestionar su inventario de productos debido a métodos manuales desorganizados, lo que genera pérdidas económicas por falta de control sobre el stock disponible, desconocimiento de los valores totales en inventario y errores en el registro de precios con y sin IVA.

Solución:
Un sistema de gestión de inventario digital basado en operaciones CRUD (Crear, Leer, Actualizar y Eliminar) que permite a los comerciantes registrar productos de forma rápida y segura, actualizar información en tiempo real, calcular automáticamente valores con y sin IVA (15%), y mantener un control preciso del stock disponible, eliminando errores humanos y optimizando la toma de decisiones comerciales.



3. Marco teórico

Python es un lenguaje de programación de alto nivel, orientado a objetos, con una semántica dinámica integrada, utilizado principalmente para el desarrollo web y de aplicaciones informáticas. Se caracteriza por poseer una sintaxis clara y sencilla, lo que facilita el aprendizaje y la lectura del código. Además, es un lenguaje interpretado, multiplataforma y de código abierto, características que lo convierten en una herramienta ampliamente utilizada en el desarrollo de software.

Un elemento que complementa a los lenguajes de alto nivel como Python son las bibliotecas. En este proyecto se utilizó Tkinter, una biblioteca gráfica que se distribuye junto con Python, por lo que no requiere instalación adicional para su uso. Tkinter permite crear interfaces gráficas de usuario (GUI), facilitando el desarrollo de aplicaciones que incluyen ventanas, botones, menús y otros elementos interactivos.

Esta biblioteca es considerada una herramienta accesible y adecuada para quienes desean iniciarse en el desarrollo de aplicaciones gráficas en Python, ya que ofrece una forma rápida y práctica de crear interfaces sin necesidad de conocimientos avanzados en diseño gráfico.

Además, en el desarrollo del sistema se implementó el concepto de persistencia de datos, el cual se refiere a la capacidad de un programa para almacenar información de manera permanente, permitiendo que los datos puedan ser recuperados y utilizados posteriormente. En Python, esta persistencia se puede lograr mediante el manejo de archivos de texto, utilizando funciones integradas del lenguaje.

La combinación del lenguaje Python, la biblioteca gráfica Tkinter y la persistencia de datos mediante archivos constituye la base de nuestro proyecto que permitió desarrollar un sistema de tienda con un CRUD básico, el cual facilita la gestión visual y el almacenamiento de información de manera estructurada.

4. Desarrollo

4.1 Variables y arreglos globales

IVA_ECUADOR: Es un variable en donde se almacena el valor del IVA (15%) que se les impone a los productos en Ecuador.

```
IVA_ECUADOR = 0.15
```

ARCHIVO: Variable que almacena el archivo de texto(.txt) llamada productos.

```
ARCHIVO = "productos.txt"
```

nombre_producto: Es una lista que de momento se la inicializa vacía.

```
nombre_producto = []
```

stock_disponible: Es una lista que de momento se la inicializa vacía.

```
stock_disponible = []
```

precio_neto: Es una lista que de momento se la inicializa vacía.

```
precio_neto = []
```

id_producto: Es una lista que de momento se la inicializa vacía.

```
id_producto = []
```

4.2 CRUD

4.2.1 Crear

menu (): Creación del menu de opciones de nuestra aplicación.

```
def menu():
    print("\n" + "="*50)
    print("        BARRIO MARKET - GESTIÓN CONTABLE")
    print("1) Registrar producto")
    print("2) Ver Inventario (Ganancia Total Valorada)")
    print("3) Buscar producto por ID")
    print("4) Actualizar producto")
    print("5) Eliminar producto")
    print("6) Salir")
```

generar_id (): Genera un id para el producto seleccionado.

```
Qodo: Test this function
def generar_id():
    return random.randint(1000, 9999)
```

guardar_datos (): Se abre al archivo (“a”) y al proceso lo nombramos con f, que sera el que recibira y agregara los paramatros id_p, nombre, precio, stock escribiendolos en el archivo y los guardara en la variable ARCHIVO.

```
Qodo: Test this function
def guardar_datos(id_p, nombre, precio, stock):
    with open(ARCHIVO, "a", encoding="utf-8") as f:
        f.write(f"{id_p},{nombre},{precio},{stock}\n")
```

registrar_producto (): En esta función vamos a registrar el nombre del producto, precio neto y el stock almacenándose en las variables nombre, precio, stock respectivamente.

```
Qodo: Test this function
def registrar_producto():
    try:
        nombre = input("Nombre del producto: ").strip()
        if nombre == "":
            print("El nombre no puede estar vacío")
            return

        precio = float(input("Precio neto: "))
        if precio <= 0:
            print("El precio debe ser mayor que 0")
            return

        stock = int(input("Stock disponible del producto: "))
        if stock < 0:
            print("El stock no puede ser negativo")
            return

    except ValueError:
        print("Precio o stock inválido")
        return
```

Una vez el usuario haya ingresado los datos, estos se añaden a su respectiva lista con el método append. El único dato el cual el usuario no lo ingresa es el id del producto ya que eso se lo asignamos nosotros con la función generar_id (), por último, guardamos la información con la función guardar datos ().

```
    except ValueError:
        print("Precio o stock inválido")
        return

    # Generar ID
    id_p = generar_id()

    # Guardar en listas
    id_producto.append(id_p)
    nombre_producto.append(nombre)
    precio_neto.append(precio)
    stock_disponible.append(stock)

    # Guardar en archivo
    guardar_datos(id_p, nombre, precio, stock)

    print("\n Producto registrado exitosamente")
    print(f"ID del producto: {id_p}")
```

4.2.2 Leer

mostrar_inventario (): Esta función nos va a permitir ver y leer el inventario que tenemos en nuestra primero con el método len () verificamos que si tengamos productos de lo contrario dirá que No existen productos registrados, una vez revisado y nota que si tenemos productos procederá a calcular el valor neto y con IVA de cada producto almacenándolos en valor_producto_neto y valor_producto_iva respectivamente y una vez terminado el calculo se añaden a dos nuevas variables valor_total_neto y valor_total_con_iva haciendo que se sumen todos los valores de los productos para generar un valor total neto y un total con IVA .

```
Qodo: Test this function
def mostrar_Inventario():
    print("\n INVENTARIO ")
    print("=" * 50)

    if len(id_producto) == 0:
        print("No existen productos registrados.")
        return

    valor_total_neto = 0
    valor_total_con_iva = 0

    for i in range(len(id_producto)):
        # Cálculos por producto
        valor_producto_neto = precio_neto[i] * stock_disponible[i]
        valor_producto_iva = valor_producto_neto * (1 + IVA_ECUADOR)

        # Sumar a los totales generales
        valor_total_neto += valor_producto_neto
        valor_total_con_iva += valor_producto_iva
```

Para finalizar mostramos la información imprimiendo las variables donde almacenamos nuestros productos, stock y los valores que calculamos.

```
    # Mostrar detalle (Sin precio unitario con IVA)
    print(f"ID Producto: {id_producto[i]}")
    print(f"Nombre: {nombre_producto[i]}")
    print(f"Precio neto: ${precio_neto[i]:.2f}")
    # (Se eliminó la línea de 'Precio con IVA' unitario como pediste)
    print(f"Stock: {stock_disponible[i]}")
    print(f"Valor Neto en Stock: ${valor_producto_neto:.2f}")
    print("-" * 50)

# Mostrar los totales acumulados al final
print(f"💰 RESUMEN FINANCIERO:")
print(f"   Valor Total (Neto):      ${valor_total_neto:.2f}")
print(f"   Valor Total (Con IVA):   ${valor_total_con_iva:.2f}")
print("=" * 50)
```

4.2.3 Actualizar

actualizar_archivo_completo (): Con esta función podremos abrir y escribir en nuestro archivo(“w”) podemos escribir justo donde necesitemos si le pasamos como parámetro el id del producto el cual vamos a sobrescribir, una vez lea la posición se actualizará con los datos que se enviaran previamente.

```
Qodo: Test this function
def actualizar_archivo_completo():
    with open(ARCHIVO, "w", encoding="utf-8") as f:
        for i in range(len(id_producto)):
            f.write(f"{id_producto[i]},{nombre_producto[i]},{precio_neto[i]},{stock_disponible[i]}\n")
```

actualizar_producto (): Esta función nos permitirá actualizar datos de cualquier producto que queramos cambiar, primero verificamos que tengamos productos de no tenerlo nos dirá El inventario está vacío, una vez verificado y ver que tenemos productos disponibles procede a preguntarle al usuario el id del producto el cual desea actualizar los datos, si encuentra el id del producto el usuario podrá añadir el nuevo precio neto y el stock que estará disponible una vez ingresados actualizamos todo el archivo con la función actualizar_archivo_completo, pero de no encontrar el id del producto que se desea actualizar nos dará un error el cual dirá que no se encontró tal producto

```
Qodo: Test this function
def actualizar_producto():
    try:
        if len(id_producto) == 0:
            print("El inventario está vacío.")
            return

        codigo = int(input("\nIngrese el ID para editar: "))
        for i in range(len(id_producto)):
            if codigo == id_producto[i]:
                print(f"Modificando: {nombre_producto[i]}")

                nuevo_p = input(f"Nuevo precio neto [Actual: ${precio_neto[i]}]: ")
                if nuevo_p:
                    precio_neto[i] = float(nuevo_p)

                nuevo_s = input(f"Nuevo stock [Actual: {stock_disponible[i]}]: ")
                if nuevo_s:
                    stock_disponible[i] = int(nuevo_s)

                actualizar_archivo_completo()

                print("✅ Datos actualizados correctamente.")
                return
        print("❌ Error: ID no encontrado.")
    except ValueError:
        print("❌ Error: Datos ingresados no válidos.")
```

4.2.4 Eliminar

eliminar_producto (): Con esta función podremos eliminar cualquier producto que deseemos, nuevamente revisara si tenemos productos en el inventario de no tenerlo nos dirá EL inventario está vacío, una vez verificado y ver que tenemos productos nos pedirá ingresar el id del producto que deseamos eliminar, si se nos ocurre escribir un carácter en vez de un número este nos hará saber que debemos enviar un número, el sistema al encontrar el producto nos desplegara la información respectiva del producto y nos preguntara si estamos seguros de eliminarlo.

```
Qodo: Test this function
def eliminar_producto():
    if len(id_producto) == 0:
        print("El inventario está vacío.")
        return

    try:
        id_eliminar = int(input("Ingrese el ID del producto a eliminar: "))
    except ValueError:
        print("❌ El ID debe ser un número.")
        return

    eliminado = False

    for i in range(len(id_producto)):
        if id_producto[i] == id_eliminar:
            print("\nProducto encontrado:")
            print("--------------------------------")
            print("ID:", id_producto[i])
            print("Nombre:", nombre_producto[i])
            print("Precio:", precio_neto[i])
            print("Stock:", stock_disponible[i])
            print("--------------------------------")

            confirmacion = input("\n¿Está seguro de eliminar este producto? (S/N): ").upper()
```

De estar seguros y querer eliminar el producto se presiona S (si) esto ara que se elimine todo dato de dicho producto y se utilizara una función necesaria del actualizar actualizar_archivo_producto () para guardar este cambio y una vez realizado nos dirá Producto eliminado correctamente, pero de haber elegido no (N) se romperá la acción de eliminar y nos mostrara Eliminación cancelada, de ultimo si no se encontró el id ingresado dirá que no existe y no entrara a ningún proceso antes mencionado

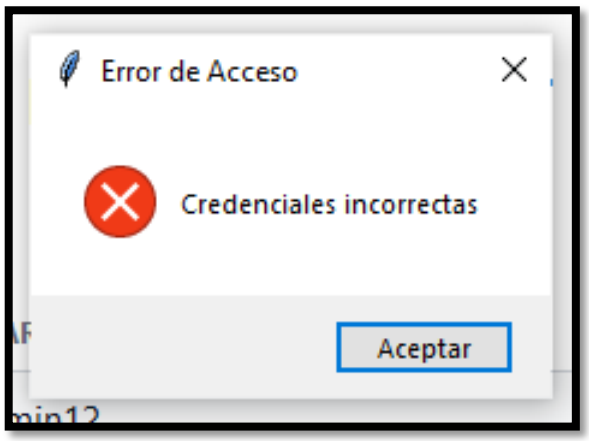
:

```
            if confirmacion == "S":
                id_producto.pop(i)
                nombre_producto.pop(i)
                precio_neto.pop(i)
                stock_disponible.pop(i)
                actualizar_archivo_completo()
                print("Producto eliminado correctamente.")
            else:
                print("Eliminación cancelada.")

            eliminado = True
            break

    if not eliminado:
        print("El ID ingresado no existe.")
```

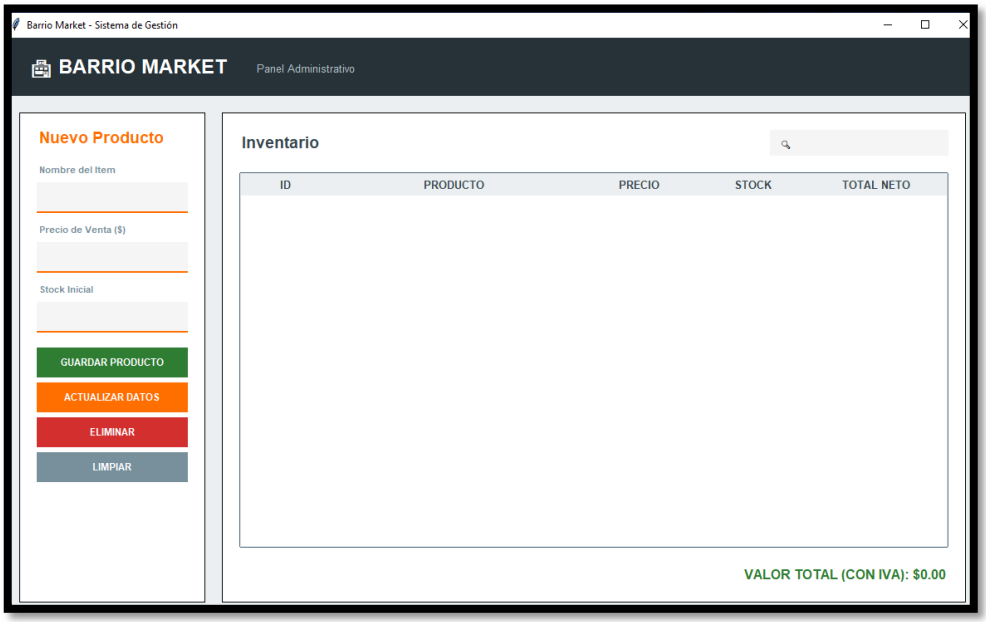
5.Manual de usuario

Login: En esta interfas encontrara un cordial mensaje de biendenida y la imagen de nuestra empresa, a contiacion observa dos campos uno de usuario y otro de contraseña aquí debera ingresar sus respectivas credenciales.

De no hacerlo le aparecera una mensaje informandole que las credenciales que ingreso son incorrectas y debe ingresar credenciales validas.



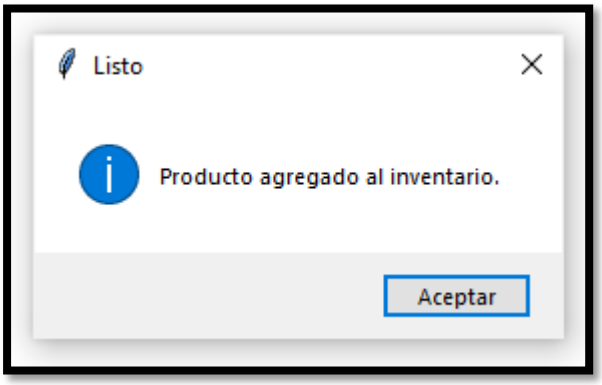
Pantalla principal: Una vez haya digitado las credenciales correctas, el sistema lo llevará a nuestra pantalla principal, donde podrá observar el espacio en el que podrá escribir y realizar las acciones de un CRUD, como lo son Crear, Leer, Actualizar y Eliminar, con los diferentes productos que desee de una tienda.



Crear un porducto: Para crear un producto usted primer debera escribir la informacion del mismo en el panel de la izquiera que dice “NUEVO PRODUCTO”

Una vez escrita la información, solo debe presionar el botón de color verde que dice “GUARDAR PRODUCTO” y su producto se guardará automáticamente.

Le aparecerá un mensaje de confirmación que le hará saber que su producto fue guardado exitosamente.



También podrá observar que el producto ya aparece en la pantalla principal del inventario.

Y no solo eso, usted podrá ver cuál es el precio total de dicho producto que tiene en stock, con IVA (15%) y sin IVA.

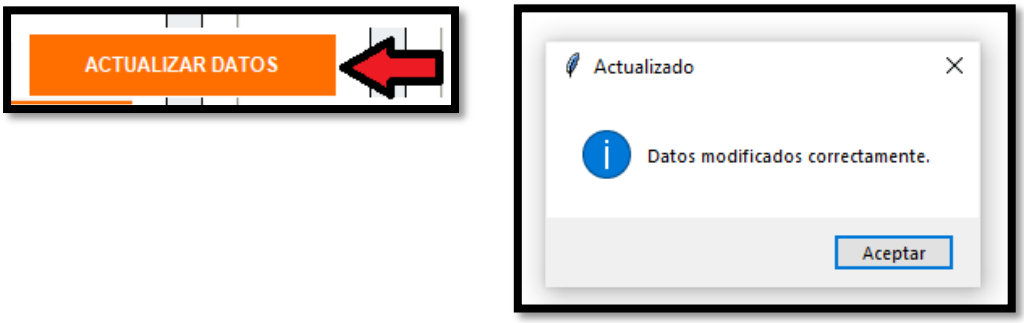
Como observación, si trata de ingresar caracteres donde deberían ir números (precio de venta y stock inicial), o viceversa, trata de ingresar números donde

deberían ir caracteres (nombre del ítem), el sistema le dará un error indicándole que verifique que los datos estén bien escritos.

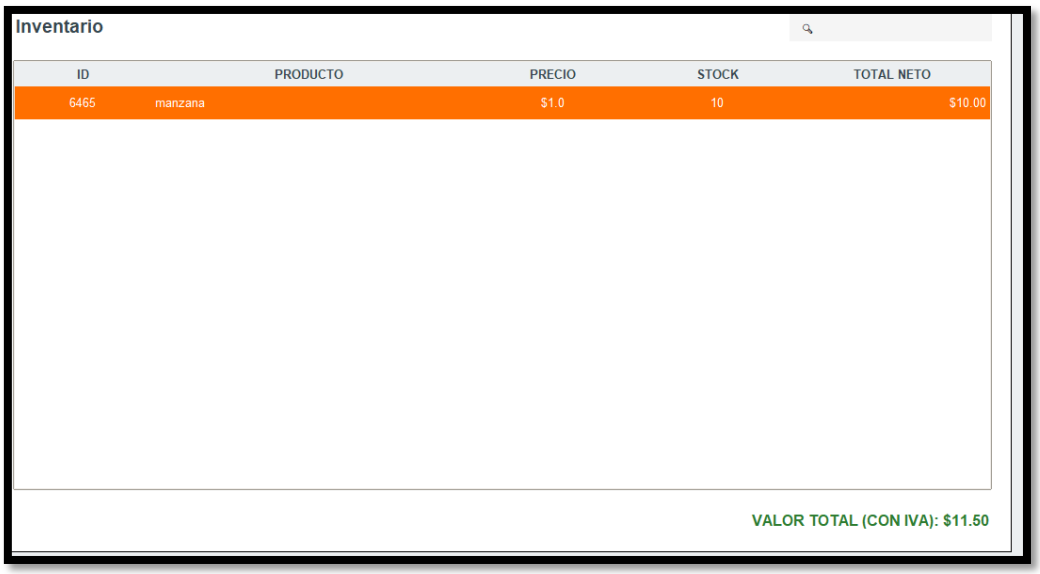
Actualizar datos: Para la actualización, solo debemos seleccionar el producto al cual queramos cambiar los datos. Al seleccionarlo, este se marcará de color naranja y en el lado izquierdo aparecerán los datos que podemos modificar.

Una vez haya hecho los cambios, solo debe presionar el botón de color naranja que dice “ACTUALIZAR DATOS” y automáticamente los cambios se guardarán

y aparecerá un cuadro de texto que dirá que los datos han sido modificados con éxito.

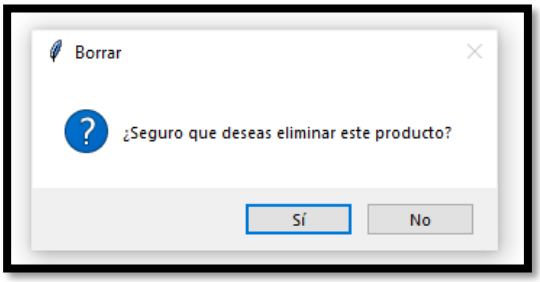


Podra observar los cambios que realizo en la pantalla principal del sistema.



Eliminar Datos: Para eliminar datos al igual que el actualizar debemos seleccionar cual es el producto que deseamos eliminar y una vez seleccionado se marcara de color naranja.

Una vez elegido, solo nos queda presionar el botón 'ELIMINAR', que al presionarlo hará que nos muestre un mensaje de confirmación preguntando si estamos seguros de eliminar dicho producto.



Al presionar “No”, el proceso de eliminación de producto se cancelará, pero de elegir “Sí”, este hará que se elimine automáticamente el producto.

6. Conclusiones

- Se logró implementar un sistema CRUD eficiente mediante Python y Tkinter, optimizando la gestión de inventario y garantizando una experiencia de usuario fluida y funcional.

- Logramos crear un CRUD con varias validaciones para que el programa sea seguro tanto en el guardado e ingreso de datos, haciendo que sea la información correcta y clara cuando se le muestre al usuario, además el uso de python y tkinder a sido fundamental haciendo una pieza fundamental en este proyecto.

- El desarrollo de un sistema CRUD de productos utilizando Python y Tkinter permitió comprender y aplicar conceptos fundamentales de la programación orientada a objetos, manejo de datos y diseño de interfaces gráficas. Gracias a Tkinter, se logró crear una interfaz amigable e intuitiva que facilita la interacción del usuario con el sistema, permitiendo crear, leer, actualizar y eliminar productos de manera eficiente.

- Logramos una persistencia de datos sin necesidad de una base de datos usando archivos en su lugar los cuales podemos leer, editar, eliminar, sobreescribir, etc gracias a las funciones que nos brinda python para archivos

- El proyecto de inventario tipo market con la implementación de un CRUD nos permitió aplicar en la práctica los conceptos básicos de algoritmos, como el uso de estructuras de datos, condicionales, ciclos y funciones. Mediante las operaciones de crear, leer, actualizar y eliminar productos, pudimos entender mejor cómo funciona la lógica de un sistema. En conclusión, este proyecto ayudó a mejorar nuestro razonamiento lógico y nos dio una base para desarrollar programas más complejos en el futuro.

Extra

Trabajos Futuros

Como trabajo futuro, el sistema desarrollado puede ser mejorado y ampliado a medida que se adquieran nuevos conocimientos en semestres superiores. Entre las posibles mejoras se encuentra la implementación de una base de datos en lugar del uso de archivos de texto, lo que permitiría un manejo más eficiente y seguro de la información. Asimismo, se podría optimizar la interfaz gráfica, incorporando diseños más avanzados y una mejor experiencia de usuario. Estas mejoras permitirían fortalecer la funcionalidad y escalabilidad de la aplicación, manteniendo la estructura del CRUD implementado en el proyecto actual.