



ALGORITMO Y ESTRUCTURA DE DATOS

Integrantes:

Mateo Rodriguez

Ivonne Chauca

David Guato

Sebastián Villagomez

Dennes Molina

Diego Toapanta



PROBLEMATICA

La mayoría de tiendas pequeñas en barrios pequeños sufren de invisibilidad operativa y financiera ya que al depender de métodos manuales o de la memoria. Sin un sistema digital, el comerciante enfrenta el "caos del cuaderno", donde la incertidumbre sobre el stock real provoca ventas perdidas y el desconocimiento del valor total de la inversión impide una toma de decisiones inteligente.



ALCANCE DEL PROYECTO

Este proyecto busca solventar la problemática común en barrios de bajos recursos con una gestión de tiendas un poco obsoletas el programa que hicimos cubre todo el ciclo de vida de los datos: desde la captura y validación de productos con cálculo automático de impuestos (IVA 15%), pasando por el almacenamiento persistente en archivos de texto para garantizar que la información no se pierda, hasta la generación de reportes financieros en tiempo real que valorizan el capital invertido así ya no hace falta el uso de cuadernos o tu memoria que son poco eficientes, si no usamos este programa que es mucho mas cómodo, eficiente, útil y intuitivo



IMPLEMENTACIONES DEL PROYECTO

Sistematizar el control de inventarios: Desarrollamos una herramienta que permita el registro, búsqueda, actualización y eliminación de productos de forma eficiente.

Automatizar cálculos financieros: Implementamos un sistema que calcule automáticamente el valor neto y el valor con IVA (15%) de los productos en stock para facilitar el balance contable de la tienda.

Implementar persistencia de datos: Utilizamos el módulo os y manejo de archivos en Python para leer y escribir información en disco de forma dinámica.

HERRAMIENTAS QUE USAMOS



Curso de Tkinter: Crea Interfaces Gráficas en Python Desde Cero Hasta Avanzado

Código Espinoza - Automatiza tu Vida · Curso

Curso de Tkinter de Python: Crea Tu Primera Ventana | Aprende a Diseñar Interfaces Gráficas | E01 · 3:38

Python Tkinter Curso Completo | Crea GUI Profesionales con Python | E02 · 12:11

[Ver curso completo](#)



Python 3.14.2 documentation

Welcome! This is the official documentation for Python 3.14.2.

TKINTER

Es la herramienta que te permite empezar a crear ventanas con botones, menús, cuadros de texto e imágenes, muy similares a las aplicaciones que usas en Windows, MacOS o Linux.

¿PORQUE USAMOS TKINTER?

Viene incluido: No necesitas instalar nada externo; si tienes Python, ya tienes Tkinter.

Multiplataforma: El mismo código que escribes funciona en cualquier sistema operativo.

Basado en Widgets: Todo en Tkinter es un "Widget" (un componente visual).

BARRIO MARKET

Tienda

Nuestra aplicación, Barrio Market, es una solución ligera para pequeños negocios que necesitan digitalizar su inventario. No solo permite organizar productos, sino que automatiza el cálculo de impuestos y asegura la integridad de los datos mediante validaciones técnicas y almacenamiento persistente.



PARTES DEL CRUD

CREAR

Esta función `registrar_producto` es el punto de entrada de datos y el motor que asegura que la información de la tienda sea coherente y real. Lo más importante aquí es la validación y limpieza de datos.

Antes de guardar cualquier cosa, nuestro programa verifica que el nombre no esté vacío y que el precio y el stock sean números lógicos (mayores a cero). Esto evita que el sistema falle más adelante con cálculos matemáticos imposibles, actuando como un filtro de seguridad.

```
import random

IVA_ECUADOR = 0.15
ARCHIVO = "productos.txt"

id_producto = []
nombre_producto = []
precio_neto = []
stock_disponible = []

def generar_id():
    return random.randint(1000, 9999)

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")

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

    # 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}")

registrar_producto()
```

PARTES DEL CRUD

LEER

La función `mostrar_Inventario` es el rol como módulo de inteligencia financiera, ya que transforma los datos crudos almacenados en información útil para la toma de decisiones. No se limita a listar nombres, sino que realiza un procesamiento en tiempo real para calcular el impacto impositivo (IVA) y la valorización del capital invertido en stock.

```
def mostrar_Inventario():

    print("\n INVENTARIO ")
    print("-" * 50)

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

    valor_total = 0

    for i in range(len(id_producto)):
        precio_con_iva = precio_neto[i] * (1 + IVA_ECUADOR)
        valor_producto = precio_neto[i] * stock_disponible[i]
        valor_total += valor_producto

        print(f"ID Producto: {id_producto[i]}")
        print(f"Precio neto: ${precio_neto[i]:.2f}")
        print(f"Precio con IVA: ${precio_con_iva:.2f}")
        print(f"Stock: {stock_disponible[i]}")
        print(f"Valor en inventario: ${valor_producto:.2f}")
        print("-" * 50)

    print(f" Valor total del inventario: ${valor_total:.2f}")
```

PARTES DEL CRUD

ACTUALIZAR

La función `actualizar_producto` tiene flexibilidad operativa, ya que permite modificar selectivamente los atributos clave de un producto (precio y stock) sin obligar al usuario a reescribir toda la información. Implementamos una lógica de búsqueda por ID y condicionales internos que validan si el usuario dejó el campo vacío o ingresó un nuevo valor, la función optimiza el mantenimiento del inventario; esto garantiza la integridad de la información y evitan errores de duplicidad

```
def actualizar_producto():
    try:
        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)

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

PARTES DEL CRUD

ELIMINAR

La función `eliminar_producto` implementamos un mecanismo de seguridad de doble confirmación, el cual actúa como una barrera contra errores accidentales que podrían causar la pérdida de datos, la función destaca por su precisión en la gestión de memoria y sincronización, ya que utiliza el método `.pop(i)` para eliminar simultáneamente el elemento de todas las listas paralelas basadas en el índice del producto encontrado. Además, al finalizar el proceso, invoca a `actualizar_archivo_completo()`, lo que garantiza que el cambio se refleje de inmediato en el almacenamiento físico

```
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("X 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()

            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.")
```



CONCLUSIONES Y RECOMENDACIONES

En conclusión: el proyecto demuestra que la automatización mediante Python no solo optimiza el tiempo de gestión, sino que elimina la incertidumbre financiera al transformar un inventario físico en un reporte digital exacto y persistente.

Recomendación: Implementar cuadros de diálogo de confirmación (messagebox) para acciones críticas como eliminar productos y añadir validaciones en tiempo real que impidan ingresar letras en campos de precio también implementar funciones para evitar que la aplicación se cierre inesperadamente, y utilizar un componente de tabla (Treeview) para que el inventario se visualice de forma organizada y moderna, facilitando la lectura de los datos .

FEBRERO 2

BARRIO MARKET

ECUADOR



Muchas
GRACIAS