UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

UNIVERSIDAD ESTATAL AMAZÓNICA

INGENIERÍA EN TECNOLOGÍAS DE INFORMACIÓN

PROGRAMACIÓN ORIENTADA A OBJETOS

TEMA:

SISTEMA AVANZADO DE GESTIÓN DE INVENTARIO

ESTUDIANTE:

EDWIN FABIÁN NORIEGA BALDEON

DOCENTE:

ING. SANTIAGO ISRAEL NOGALES GUERRERO

NIVEL:

SEGUNDO NIVEL_PARALELO “A”

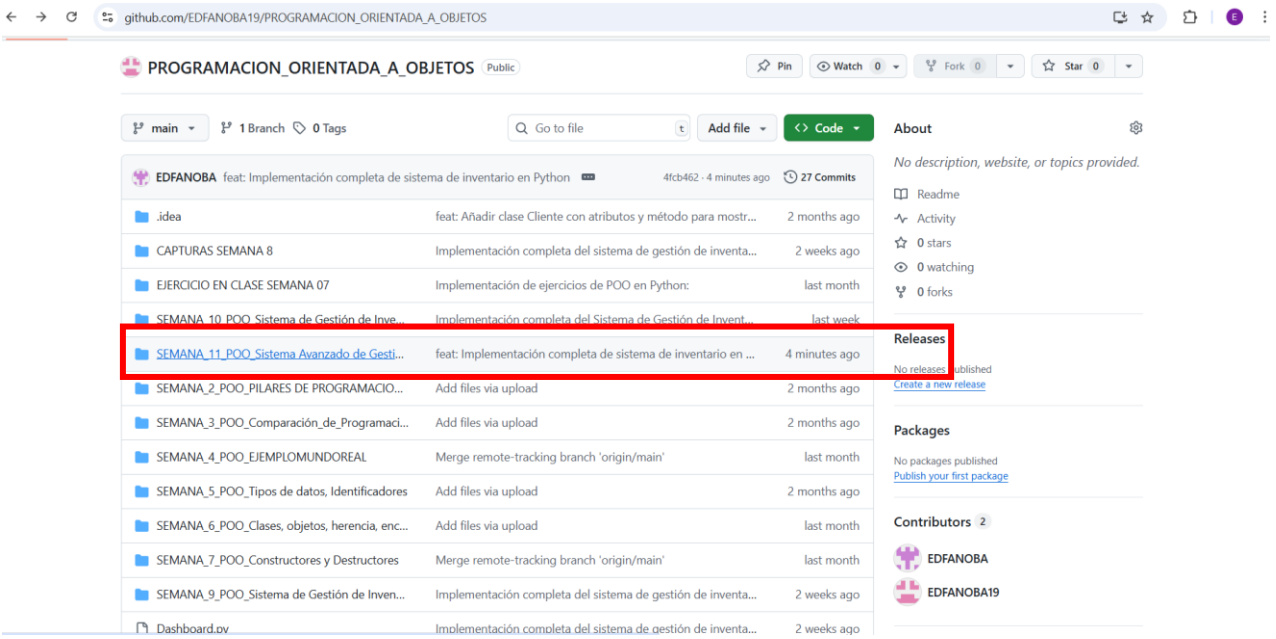
2025-2025

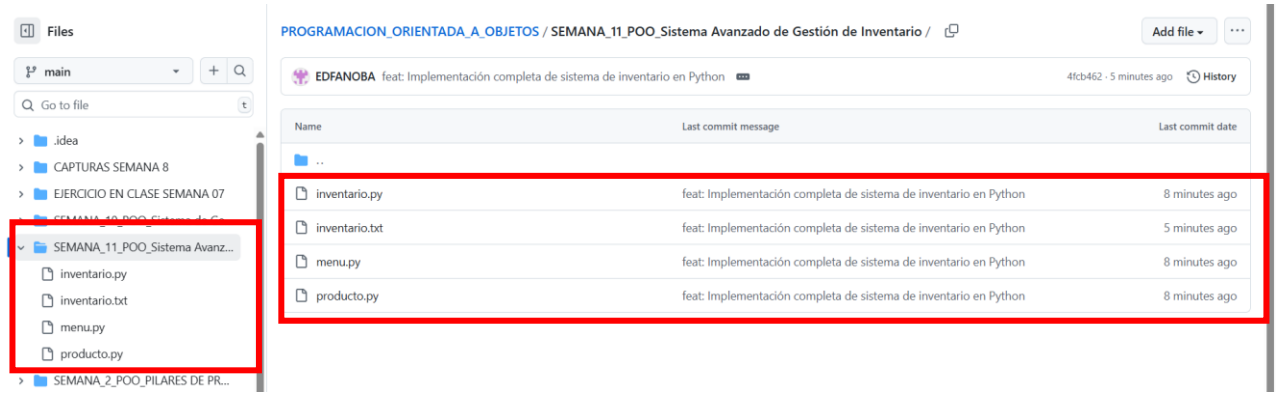
PUYO-ECUADOR

1. **CÓDIGO DE GITHUB**

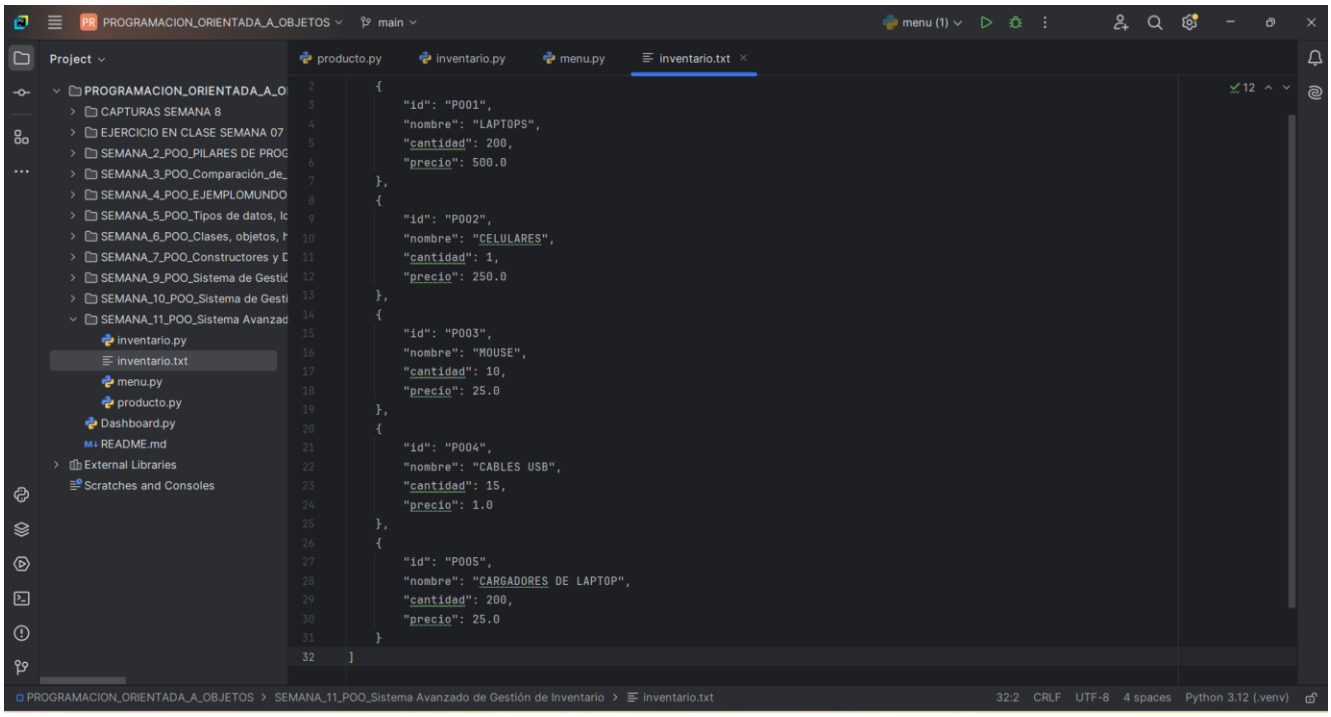
https://github.com/EDFANOBA19/PROGRAMACION_ORIENTADA_A_OBJETOS.git

2. **REPOSITORIO DE GITHUB**

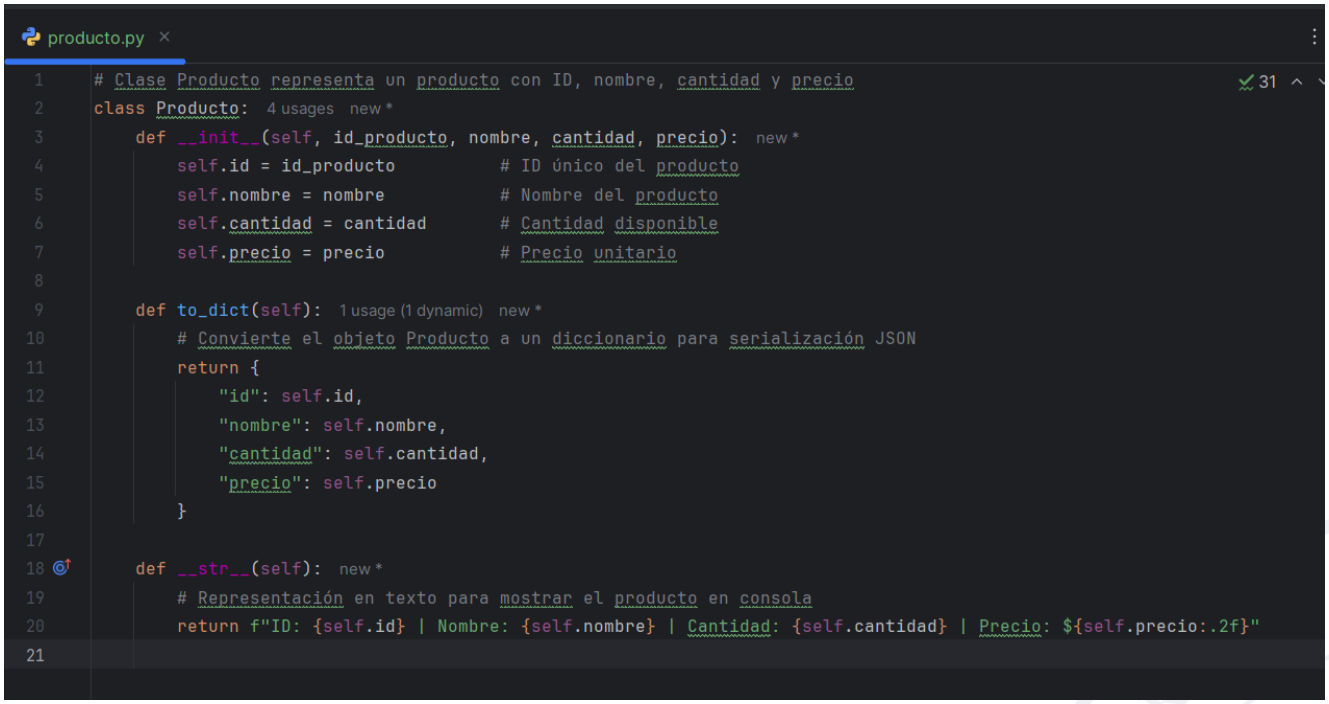




2.1 DESARROLLO DE CÓDIGO CON SUS DIFERENTES PEDIDOS Y REQUISITOS



2.1.1 Clase Producto: Debe contener atributos como ID (único), nombre, cantidad y precio. Implementa métodos para obtener y establecer estos atributos.



2.1.2 Clase Inventario: Debe utilizar una colección adecuada (p. ej., un diccionario) para almacenar los productos. Implementa métodos para:

- Añadir nuevos productos.
- Eliminar productos por ID.

- Actualizar cantidad o precio de un producto.
- Buscar y mostrar productos por nombre.
- Mostrar todos los productos en el inventario.

inventario.py

```python
import json
import os
from producto import Producto  # Importa la clase Producto

# Clase Inventario maneja la colección de productos y el almacenamiento en archivo
class Inventario:  2 usages  new *
    def __init__(self, archivo='inventario.txt'):  new *
        self.archivo = archivo          # Archivo donde se guarda el inventario
        self.productos = {}             # Diccionario de productos: clave=ID, valor=Producto
        self.cargar_desde_archivo()    # Carga productos desde archivo al iniciar

    def cargar_desde_archivo(self):  1 usage  new *
        # Si archivo no existe, crea uno vacío para comenzar
        if not os.path.exists(self.archivo):
            self.productos = {}
            self.guardar_en_archivo()
            return
        try:
            with open(self.archivo, 'r', encoding='utf-8') as f:
                contenido = f.read().strip()
                # Si archivo vacío, iniciar inventario vacío
                if not contenido:
                    self.productos = {}
                else:
                    # Cargar lista JSON y convertir a diccionario de Productos
                    lista_dicts = json.loads(contenido)
                    self.productos = {p['id']: Producto(p['id'], p['nombre'], p['cantidad'], p['precio']) for p in lista_dicts}
        except (json.JSONDecodeError, FileNotFoundError):
            # Si hay error lectura, crear inventario vacío para evitar fallo
            self.productos = {}
            self.guardar_en_archivo()

    def guardar_en_archivo(self):  5 usages  new *
        # Convierte los Productos a diccionarios y guarda como JSON indentado en archivo texto
        lista_dicts = [p.to_dict() for p in self.productos.values()]
        with open(self.archivo, 'w', encoding='utf-8') as f:
            json.dump(lista_dicts, f, indent=4)

    def añadir_producto(self, producto):  1 usage  new *
        # Agrega un nuevo producto si su ID no existe
        if producto.id in self.productos:
            raise KeyError(f"Producto con ID '{producto.id}' ya existe.")
        self.productos[producto.id] = producto
        self.guardar_en_archivo()

    def eliminar_producto(self, id_producto):  1 usage  new *
        # Elimina producto por su ID si existe
        if id_producto in self.productos:
            del self.productos[id_producto]
            self.guardar_en_archivo()
        else:
            raise KeyError(f"No existe producto con ID '{id_producto}'.")

    def actualizar_producto(self, id_producto, cantidad=None, precio=None):  1 usage  new *
        # Actualiza cantidad o precio del producto indicado
        if id_producto not in self.productos:
            raise KeyError(f"No existe producto con ID '{id_producto}'.")
        if cantidad is not None:
            self.productos[id_producto].cantidad = cantidad
        if precio is not None:
            self.productos[id_producto].precio = precio
        self.guardar_en_archivo()
```

```
63
64      def buscar_por_nombre(self, nombre):  1 usage  new *
65          # Busca productos cuyo nombre contiene el texto indicado (case insensitive)
66          nombre_lower = nombre.lower()
67          return [p for p in self.productos.values() if nombre_lower in p.nombre.lower()]
68
69      def mostrar_todos(self):  1 usage  new *
70          # Muestra todos los productos presentes en el inventario
71          if not self.productos:
72              print("Inventario vacío.")
73          else:
74              print("Productos en inventario:")
75              for p in self.productos.values():
76                  print(p)
77
```

2.1.2.1 Menú

```
1   from producto import Producto
2   from inventario import Inventario
3
4   def menu():  1 usage  new *
5       inventario = Inventario()  # Crear instancia de inventario
6       while True:
7           print("\nSistema de Gestión de Inventario")
8           print("1. Añadir nuevo producto")
9           print("2. Eliminar producto por ID")
10          print("3. Actualizar cantidad o precio de un producto")
11          print("4. Buscar productos por nombre")
12          print("5. Mostrar todos los productos")
13          print("0. Salir")
14          opcion = input("Seleccione opción: ").strip()
15
16          try:
17              if opcion == '1':
18                  id_prod = input("ID (único): ").strip()
19                  nombre = input("Nombre: ").strip()
20                  cantidad = int(input("Cantidad: "))
21                  precio = float(input("Precio: "))
22                  p = Producto(id_prod, nombre, cantidad, precio)
23                  inventario.añadir_producto(p)
24                  print("Producto añadido exitosamente.")
25              elif opcion == '2':
26                  id_prod = input("ID del producto a eliminar: ").strip()
27                  inventario.eliminar_producto(id_prod)
28                  print("Producto eliminado con éxito.")
29              elif opcion == '3':
30                  id_prod = input("ID del producto a actualizar: ").strip()
31                  cantidad_input = input("Nueva cantidad (deje vacío para no modificar): ").strip()
32                  precio_input = input("Nuevo precio (deje vacío para no modificar): ").strip()
```

```
                cantidad = int(cantidad_input) if cantidad_input else None
                precio = float(precio_input) if precio_input else None
                inventario.actualizar_producto(id_prod, cantidad, precio)
                print("Producto actualizado.")
            elif opcion == '4':
                nombre = input("Nombre o parte del nombre para buscar: ").strip()
                resultados = inventario.buscar_por_nombre(nombre)
                if resultados:
                    print(f"Se encontraron {len(resultados)} productos:")
                    for p in resultados:
                        print(p)
                else:
                    print("No se encontraron productos.")
            elif opcion == '5':
                inventario.mostrar_todos()
            elif opcion == '0':
                print("Saliendo del programa.")
                break
            else:
                print("Opción inválida, intente de nuevo.")
        except ValueError:
            print("Error: cantidad y precio deben ser números válidos.")
        except KeyError as e:
            print(e)

if __name__ == "__main__":
    menu()

```

2.1.3. Integración de Colecciones: Utiliza colecciones para optimizar las operaciones del inventario, como búsqueda rápida de productos y manejo eficiente de los datos.

```
def menu():  1 usage  new *
    inventario = Inventario()  # Crear instancia de inventario
    while True:
        print("\nSistema de Gestión de Inventario")
        print("1. Añadir nuevo producto")
        print("2. Eliminar producto por ID")
        print("3. Actualizar cantidad o precio de un producto")
        print("4. Buscar productos por nombre")
        print("5. Mostrar todos los productos")
        print("0. Guardar y Salir")
        opcion = input("Seleccione opción: ").strip()

        elif opcion == '4':
            nombre = input("Nombre o parte del nombre para buscar: ").strip()
            resultados = inventario.buscar_por_nombre(nombre)
            if resultados:
                print(f"Se encontraron {len(resultados)} productos:")
                for p in resultados:
                    print(p)
            else:
                print("No se encontraron productos.")
```

2.1.4. **Almacenamiento en Archivos:** Implementa funciones para guardar y cargar el inventario desde archivos. Esto incluye la serialización de la colección del inventario para su almacenamiento y la deserialización al cargar el programa.

```
def menu():  1 usage  new *
    inventario = Inventario()  # Crear instancia de inventario
    while True:
        print("\nSistema de Gestión de Inventario")
        print("1. Añadir nuevo producto")
        print("2. Eliminar producto por ID")
        print("3. Actualizar cantidad o precio de un producto")
        print("4. Buscar productos por nombre")
        print("5. Mostrar todos los productos")
        print("0. Guardar y Salir")
        opcion = input("Seleccione opción: ").strip()

                elif opcion == '0':
                    print("Guardando y Saliendo del programa.")
                    break
                else:
                    print("Opción inválida, intente de nuevo.")
            except ValueError:
                print("Error: cantidad y precio deben ser números válidos.")
            except KeyError as e:
                print(e)
```

2.1.5. **Interfaz de Usuario:** Crea un menú interactivo en la consola que permita al usuario realizar operaciones sobre el inventario (añadir, eliminar, actualizar, buscar, mostrar).

```
Run    menu (1)
Sistema de Gestión de Inventario
1. Añadir nuevo producto
2. Eliminar producto por ID
3. Actualizar cantidad o precio de un producto
4. Buscar productos por nombre
5. Mostrar todos los productos
0. Guardar y Salir
Seleccione opción:
```

3. INSTRUCCIONES DE LA TAREA:

- Utiliza PyCharm para desarrollar tu solución. Asegúrate de que tu código esté bien organizado y comentado adecuadamente para explicar la lógica detrás de tus decisiones de diseño.

3.1 CÓDIGO COMPLETO

3.1.1 Class Producto

producto.py × inventario.py menu.py inventario.txt

```python
# Clase Producto representa un producto con ID, nombre, cantidad y precio
class Producto:  4 usages  new *
    def __init__(self, id_producto, nombre, cantidad, precio):  new *
        self.id = id_producto          # ID único del producto
        self.nombre = nombre           # Nombre del producto
        self.cantidad = cantidad       # Cantidad disponible
        self.precio = precio           # Precio unitario

    def to_dict(self):  1 usage (1 dynamic)  new *
        # Convierte el objeto Producto a un diccionario para serialización JSON
        return {
            "id": self.id,
            "nombre": self.nombre,
            "cantidad": self.cantidad,
            "precio": self.precio
        }

    def __str__(self):  new *
        # Representación en texto para mostrar el producto en consola
        return f"ID: {self.id} | Nombre: {self.nombre} | Cantidad: {self.cantidad} | Precio: ${self.precio:.2f}"

```

3.1.2 Class Inventario

producto.py × inventario.py × menu.py inventario.txt

```python
import json
import os
from producto import Producto  # Importa la clase Producto

# Clase Inventario maneja la colección de productos y el almacenamiento en archivo
class Inventario:  2 usages  new *
    def __init__(self, archivo='inventario.txt'):  new *
        self.archivo = archivo          # Archivo donde se guarda el inventario
        self.productos = {}             # Diccionario de productos: clave=ID, valor=Producto
        self.cargar_desde_archivo()     # Carga productos desde archivo al iniciar

    def cargar_desde_archivo(self):  1 usage  new *
        # Si archivo no existe, crea uno vacío para comenzar
        if not os.path.exists(self.archivo):
            self.productos = {}
            self.guardar_en_archivo()
            return
        try:
            with open(self.archivo, 'r', encoding='utf-8') as f:
                contenido = f.read().strip()
                # Si archivo vacío, iniciar inventario vacío
                if not contenido:
                    self.productos = {}
                else:
                    # Cargar lista JSON y convertir a diccionario de Productos
                    lista_dicts = json.loads(contenido)
                    self.productos = {p['id']: Producto(p['id'], p['nombre'], p['cantidad'], p['precio']) for p in lista_dicts}
        except (json.JSONDecodeError, FileNotFoundError):
            # Si hay error lectura, crear inventario vacío para evitar fallo
            self.productos = {}
            self.guardar_en_archivo()

```

```python
    def guardar_en_archivo(self):  5 usages  new *
        # Convierte los Productos a diccionarios y guarda como JSON indentado en archivo texto
        lista_dicts = [p.to_dict() for p in self.productos.values()]
        with open(self.archivo, 'w', encoding='utf-8') as f:
            json.dump(lista_dicts, f, indent=4)

    def añadir_producto(self, producto):  1 usage  new *
        # Agrega un nuevo producto si su ID no existe
        if producto.id in self.productos:
            raise KeyError(f"Producto con ID '{producto.id}' ya existe.")
        self.productos[producto.id] = producto
        self.guardar_en_archivo()

    def eliminar_producto(self, id_producto):  1 usage  new *
        # Elimina producto por su ID si existe
        if id_producto in self.productos:
            del self.productos[id_producto]
            self.guardar_en_archivo()
        else:
            raise KeyError(f"No existe producto con ID '{id_producto}'.")

    def actualizar_producto(self, id_producto, cantidad=None, precio=None):  1 usage  new *
        # Actualiza cantidad o precio del producto indicado
        if id_producto not in self.productos:
            raise KeyError(f"No existe producto con ID '{id_producto}'.")
        if cantidad is not None:
            self.productos[id_producto].cantidad = cantidad
        if precio is not None:
            self.productos[id_producto].precio = precio
        self.guardar_en_archivo()
```

```python
    def buscar_por_nombre(self, nombre):  1 usage  new *
        # Busca productos cuyo nombre contiene el texto indicado (case insensitive)
        nombre_lower = nombre.lower()
        return [p for p in self.productos.values() if nombre_lower in p.nombre.lower()]

    def mostrar_todos(self):  1 usage  new *
        # Muestra todos los productos presentes en el inventario
        if not self.productos:
            print("Inventario vacío.")
        else:
            print("Productos en inventario:")
            for p in self.productos.values():
                print(p)
```

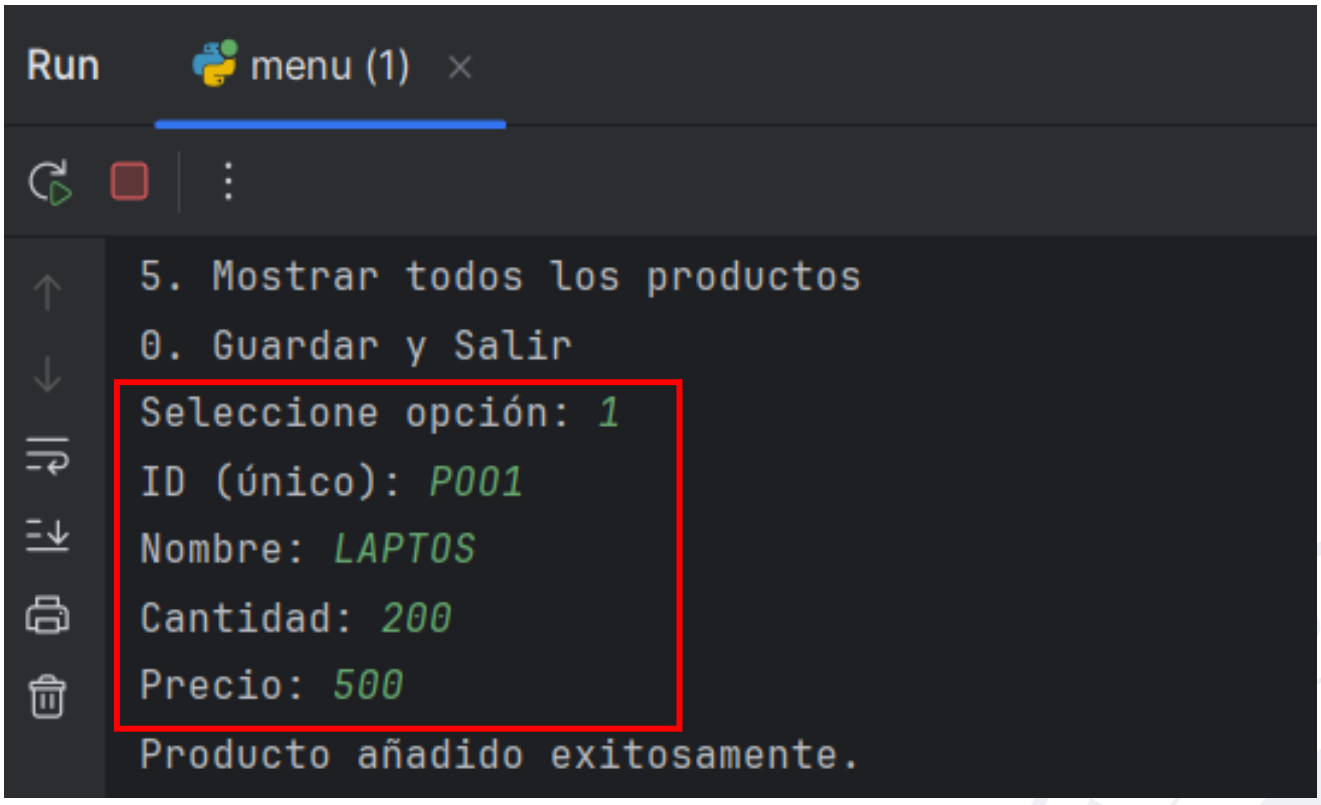
3.1.3 Menú

producto.py | inventario.py | menu.py | inventario.txt

```python
from producto import Producto
from inventario import Inventario

def menu():  1 usage  new *
    inventario = Inventario()  # Crear instancia de inventario
    while True:
        print("\nSistema de Gestión de Inventario")
        print("1. Añadir nuevo producto")
        print("2. Eliminar producto por ID")
        print("3. Actualizar cantidad o precio de un producto")
        print("4. Buscar productos por nombre")
        print("5. Mostrar todos los productos")
        print("0. Guardar y Salir")
        opcion = input("Seleccione opción: ").strip()

        try:
            if opcion == '1':
                id_prod = input("ID (único): ").strip()
                nombre = input("Nombre: ").strip()
                cantidad = int(input("Cantidad: "))
                precio = float(input("Precio: "))
                p = Producto(id_prod, nombre, cantidad, precio)
                inventario.añadir_producto(p)
                print("Producto añadido exitosamente.")
            elif opcion == '2':
                id_prod = input("ID del producto a eliminar: ").strip()
                inventario.eliminar_producto(id_prod)
                print("Producto eliminado con éxito.")
            elif opcion == '3':
                id_prod = input("ID del producto a actualizar: ").strip()
                cantidad_input = input("Nueva cantidad (deje vacío para no modificar): ").strip()
                precio_input = input("Nuevo precio (deje vacío para no modificar): ").strip()
                precio_input = input("Nuevo precio (deje vacío para no modificar): ").strip()
                cantidad = int(cantidad_input) if cantidad_input else None
                precio = float(precio_input) if precio_input else None
                inventario.actualizar_producto(id_prod, cantidad, precio)
                print("Producto actualizado.")
            elif opcion == '4':
                nombre = input("Nombre o parte del nombre para buscar: ").strip()
                resultados = inventario.buscar_por_nombre(nombre)
                if resultados:
                    print(f"Se encontraron {len(resultados)} productos:")
                    for p in resultados:
                        print(p)
                else:
                    print("No se encontraron productos.")
            elif opcion == '5':
                inventario.mostrar_todos()
            elif opcion == '0':
                print("Guardando y Saliendo del programa.")
                break
            else:
                print("Opción inválida, intente de nuevo.")
        except ValueError:
            print("Error: cantidad y precio deben ser números válidos.")
        except KeyError as e:
            print(e)

if __name__ == "__main__":
    menu()
```

- Prueba todas las funcionalidades de tu programa para asegurarte de que funcionan correctamente.
- **INVENTARIO TXT DICCIONARIO**

1. AÑADIR PRODUCTOS

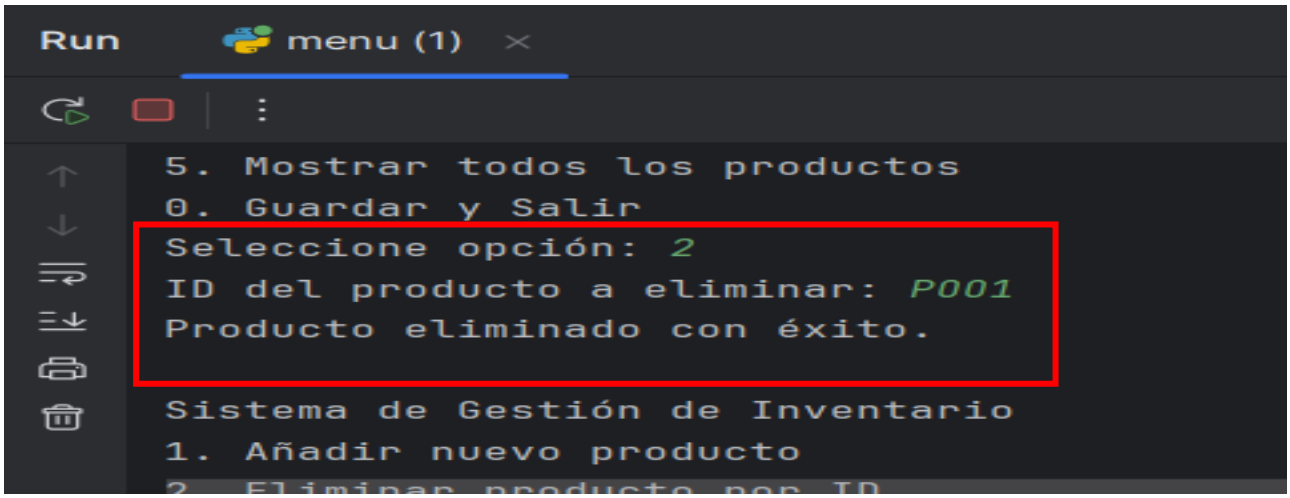


1.1 Visualización de diccionario

producto.py inventario.py menu.py inventario.txt

```
[
    {
        "id": "P001",
        "nombre": "LAPTOPS",
        "cantidad": 200,
        "precio": 500.0
    },
    {
        "id": "P002",
        "nombre": "CELULARES",
        "cantidad": 1,
        "precio": 250.0
    },
    {
        "id": "P003",
        "nombre": "MOUSE",
        "cantidad": 10,
        "precio": 25.0
    },
    {
        "id": "P004",
        "nombre": "CABLES USB",
        "cantidad": 15,
        "precio": 1.0
    },
    {
        "id": "P005",
        "nombre": "CARGADORES DE LAPTOP",
        "cantidad": 200,
        "precio": 25.0
    }
]
```

2. ELIMINAR PRODUCTO POR ID



producto.py | inventario.py | menu.py | inventario.txt

```
[
    {
        "id": "P002",
        "nombre": "CELULARES",
        "cantidad": 1,
        "precio": 250.0
    },
    {
        "id": "P003",
        "nombre": "MOUSE",
        "cantidad": 10,
        "precio": 25.0
    },
    {
        "id": "P004",
        "nombre": "CABLES USB",
        "cantidad": 15,
        "precio": 1.0
    },
    {
        "id": "P005",
        "nombre": "CARGADORES DE LAPTOP",
        "cantidad": 200,
        "precio": 25.0
    }
]
```

3. ACTUALIZAR CANTIDAD O PRECIO DE UN PRODUCTO

producto.py | inventario.py | menu.py | inventario.txt

```
[
    {
        "id": "P002",
        "nombre": "CELULARES",
        "cantidad": 1,
        "precio": 250.0
    },
    {
        "id": "P003",
        "nombre": "MOUSE",
        "cantidad": 10,
        "precio": 25.0
    },
    {
        "id": "P004",
        "nombre": "CABLES USB",
        "cantidad": 15,
        "precio": 1.0
    },
    {
```

Run | menu (1)

```
4. Buscar productos por nombre
5. Mostrar todos los productos
0. Guardar y Salir
Seleccione opción: 3
ID del producto a actualizar: P004
Nueva cantidad (deje vacío para no modificar): 25
Nuevo precio (deje vacío para no modificar): 2.00
Producto actualizado.
```

producto.py | inventario.py | menu.py | inventario.txt


```
[
    {
        "id": "P002",
        "nombre": "CELULARES",
        "cantidad": 1,
        "precio": 250.0
    },
    {
        "id": "P003",
        "nombre": "MOUSE",
        "cantidad": 10,
        "precio": 25.0
    },
    {
        "id": "P004",
        "nombre": "CABLES USB",
        "cantidad": 25,
        "precio": 2.0
    },
```


4. BUSCAR PRODUCTOS POR NOMBRE

Run | menu (1)

```
0. Guardar y Salir
Seleccione opción: 4
Nombre o parte del nombre para buscar: CELULARES
Se encontraron 1 productos:
ID: P002 | Nombre: CELULARES | Cantidad: 1 | Precio: $250.00

Sistema de Gestión de Inventario
1. Añadir nuevo producto
```

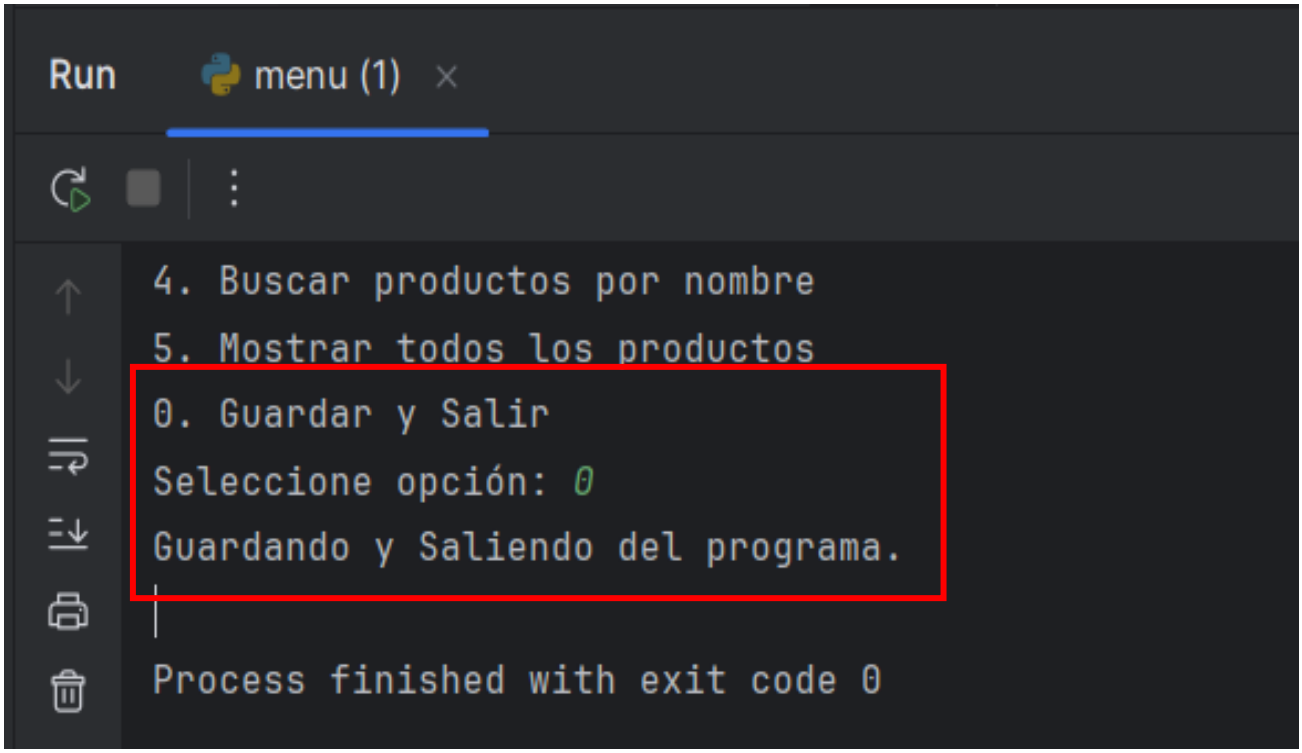
5. MOSTRAR TODOS LOS PRODUCTOS

```
Run    menu (1)
0. Guardar y Salir
Seleccione opción: 5
Productos en inventario:
ID: P001 | Nombre: LAPTOPS | Cantidad: 200 | Precio: $500.00
ID: P002 | Nombre: CELULARES | Cantidad: 1 | Precio: $250.00
ID: P003 | Nombre: MOUSE | Cantidad: 10 | Precio: $25.00
ID: P004 | Nombre: CABLES USB | Cantidad: 15 | Precio: $1.00
ID: P005 | Nombre: CARGADORES DE LAPTOP | Cantidad: 200 | Precio: $25.00
```

producto.py | inventario.py | menu.py | inventario.txt

```
[
    {
        "id": "P001",
        "nombre": "LAPTOPS",
        "cantidad": 200,
        "precio": 500.0
    },
    {
        "id": "P002",
        "nombre": "CELULARES",
        "cantidad": 1,
        "precio": 250.0
    },
    {
        "id": "P003",
        "nombre": "MOUSE",
        "cantidad": 10,
        "precio": 25.0
    },
    {
        "id": "P004",
        "nombre": "CABLES USB",
        "cantidad": 15,
        "precio": 1.0
    },
    {
        "id": "P005",
        "nombre": "CARGADORES DE LAPTOP",
        "cantidad": 200,
        "precio": 25.0
    }
]
```

6. **GUARDAR Y SALIR**



- Documenta cómo tu programa utiliza las colecciones para gestionar el inventario y cómo implementaste el almacenamiento en archivos.

Documentación del Programa de Gestión de Inventario

Uso de colecciones para gestionar el inventario

- El inventario se maneja internamente mediante un diccionario de Python.
- Este diccionario usa como clave el ID único de cada producto y como valor un objeto Producto que contiene todos los atributos (nombre, cantidad, precio).
- Gracias a que un diccionario permite acceso rápido en tiempo constante O (1) por clave, las operaciones de búsqueda, actualización y eliminación se hacen eficientes y directas.
- Para mostrar y listar productos, se itera sobre los valores del diccionario, que son objetos Producto.
- Para buscar por nombre se recorre el diccionario filtrando los productos cuyo nombre contenga el texto proporcionado, con búsqueda insensible a mayúsculas.

Esta estructura facilita la gestión dinámica y eficiente:

- No se permite duplicidad de IDs.
- Actualizar o borrar un producto es inmediato al acceder a su clave.
- Añadir un producto es simplemente asignar una nueva entrada al diccionario.

Implementación de almacenamiento en archivos

- El inventario se guarda en un archivo texto llamado inventario.txt con formato JSON legible, que es una lista de diccionarios (cada uno representa un producto).
- La serialización a JSON se hace usando:

```
lista_dicts = [p.to_dict() for p in self.productos.values()]
json.dump(lista_dicts, archivo, indent=4)
```

- La deserialización (carga desde el archivo) se hace leyendo el archivo JSON y recreando los objetos Producto desde los diccionarios con:

```
lista_dicts = json.load(archivo)
self.productos = {d["id"]: Producto(d["id"], d["nombre"], d["cantidad"], d["precio"]) for d in lista_dicts}
```

- Se incluye manejo de errores para el archivo (archivo no existente, corrupto o vacío), creándolo o inicializando el inventario en caso de problemas para asegurar que el programa funcione sin fallos.
- Cada vez que se añade, elimina o actualiza un producto, se actualiza inmediatamente el archivo JSON para mantener persistencia y evitar pérdida de datos.
- El archivo es fácilmente legible por humanos y puede editarse o consultarse manualmente si se desea.

Ventajas de este enfoque

- Usar un archivo de texto plano con JSON hace que el archivo sea fácilmente legible y portable sin necesidad de herramientas binarias específicas.
- Mantener el inventario en memoria con un diccionario permite operaciones rápidas con acceso a productos por ID.
- La estructura JSON en archivo conserva la semántica y las relaciones de los datos (lista de objetos con atributos).
- Es un formato ideal para interoperabilidad con otras aplicaciones o para migrar datos a bases de datos si el proyecto crece.
- Permite escala sencilla, ya que modificar el archivo en disco es simple y el código es mantenible.