Guía de Documentación de Código

Estándares para los Proyectos de Python

1. Introducción

En el desarrollo de software profesional, escribir código que funcione es solo la mitad del trabajo. La otra mitad es escribir código que **otros humanos (y tú mismo en el futuro) puedan entender**.

2. 1. Docstrings (Cadenas de Documentación)

Los *Docstrings* explican **QUÉ** hace una clase o una función. Se escriben entre triples comillas (") inmediatamente después de definir la función.

Estructura correcta

Un buen Docstring debe responder:

- ¿Qué hace esta función? (Resumen breve).
- ¿Qué parámetros recibe? (Si aplica).
- ¿Qué devuelve? (Si aplica).

Ejemplo Correcto

```
def calcular_descuento(precio, porcentaje):
    """
    Calcula el precio final aplicando un descuento.

    Args:
        precio (float): El precio original del producto.
        porcentaje (int): El porcentaje a descontar (0-100).

    Returns:
        float: El precio final con el descuento aplicado.
    """
    descuento = precio * (porcentaje / 100)
    return precio - descuento
```

3. 2. Comentarios (Inline Comments)

Los comentarios usan el símbolo # y sirven para explicar el **POR QUÉ** o el **CÓMO** de una parte compleja del código.

Regla de Oro: No expliques lo obvio. El código ya nos dice qué está pasando; el comentario debe decirnos por qué lo hiciste así.

Ejemplo Incorrecto (No hagas esto)

```
# Sumamos 1 a la variable contador
contador = contador + 1

# Si el precio es mayor a 100
if precio > 100:
    print("Caro")
```

¿Por qué está mal? Porque cualquiera que sepa leer Python sabe que se está sumando 1. Es ruido visual.

Ejemplo Correcto

```
# Ajustamos el contador para saltar el encabezado del archivo CSV
contador = contador + 1

# Aplicamos impuesto de lujo si supera el tope legal
if precio > 100:
    aplicar_impuesto()
```

4. 3. Ejemplo Completo para tu Proyecto

A continuación, se muestra cómo debe lucir una clase de tu proyecto.

```
class GestorInventario(QMainWindow):
    """
    Controlador principal de la ventana de Inventario.
    Gestiona la carga, guardado y visualizaci n de productos.
    """

    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.datos_productos = []

    def cargar_datos(self):
        """
        Lee el archivo JSON y rellena la tabla de productos.
        Si el archivo no existe, inicia una lista vac a.
        """
```

```
        try:
            with open("inventario.json", "r") as archivo:
                self.datos_productos = json.load(archivo)
        except FileNotFoundError:
            # Es normal la primera vez que se ejecuta la app
            self.datos_productos = []

        self.actualizar_tabla()

    def eliminar_producto(self):
        """
        Elimina la fila seleccionada tras confirmar con el usuario.
        """
        fila = self.tabla.currentRow()

        # Validaci n: Evitamos error si no hay selecci n
        if fila == -1:
            QMessageBox.warning(self, "Error", "Selecciona una fila
")
            return

        # Preguntamos antes de borrar (Buena pr ctica de UX)
        respuesta = QMessageBox.question(self, "Confirmar", "
 Borrar ?")

        if respuesta == QMessageBox.StandardButton.Yes:
            self.datos_productos.pop(fila)
            self.guardar_cambios()
            self.actualizar_tabla()
```