

MANUAL TÉCNICO
PRIMERA PRÁCTICA – LENGUAJES FORMALES Y DE PROGRAMACIÓN

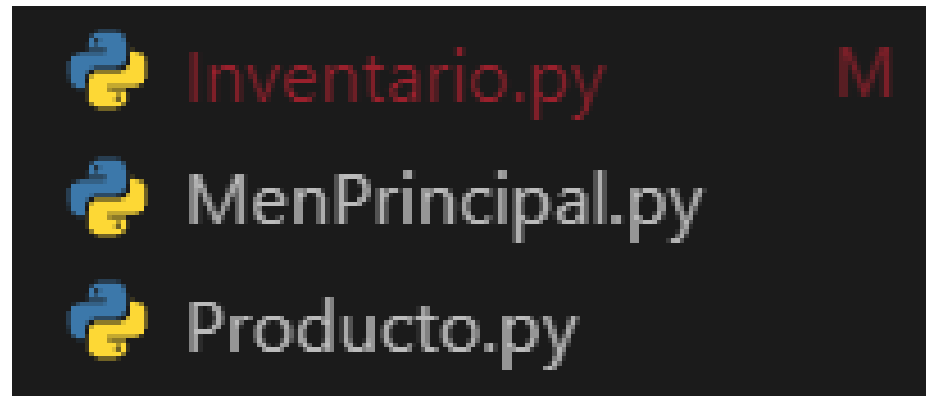
Luis Carlos Corleto Marroquín

Dirigido a:

Este Manual técnico va dirigido al auxiliar Enrique Alejandro Pinula Quiñonez.


Descripción:

Este programa permite la carga y gestión de un inventario por medio de archivos .inv y .mov para la carga del inventario y la gestión respectivamente con líneas de instrucción. El archivo .inv contendrá todos los objetos del inventario inicial con sus respectivas características como el precio, ubicación, nombre y cantidad. El archivo .mov contendrá la gestión del inventario, en este caso agregar y vender de nuevo con sus respectivas características obviando el precio unitario que se da por hecho, que el sistema realizará e identificará cada línea de instrucciones en base al producto al que le realizará la gestión. Cada tipo de gestión tiene restricciones, como por ejemplo que no se puede vender un objeto que no se tiene en el inventario o vender más de la cantidad disponible, con la instrucción de agregar, no se podrán agregar objetos que no estén en el inventario inicial, por último se puede generar informe en un archivo .txt con todos los datos actuales del inventario.

Clases:

Donde:

- **Inventario:** Clase en donde se implementan las funciones y métodos para la gestión automática, la lectura de los archivos y las verificaciones de las restricciones.
- **MenPrincipal:** Menú en donde la persona puede gestionar el inventario y los movimientos además de generar el informe.
- **Producto:** Objeto de tipo producto en donde se definen las características del producto, Nombre, cantidad, precio y ubicación:



```
class Product:
    def __init__(self, nombre, cantidad, precioUni, ubicacion):
        self.nombre = nombre
        self.cantidad = cantidad
        self.precioUni = precioUni
        self.ubicacion = ubicacion
```

Clase producto en donde se especifican las características del objeto, aquí únicamente se realiza el constructor.

```
def leer_archivo(self, nombre_archivo):
    try:
        with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
            lineas = archivo.readlines()

            for linea in lineas:
                if linea.startswith("crear_producto "):
                    partes = linea.strip().split(" ")[1].split(";")
                    if len(partes) == 4:
                        nombre, cantidad, precioUni, ubicacion = partes
                        producto = Product(nombre, int(cantidad), float(precioUni),
ubicacion)
                        self.productos.append(producto)
                    else:
                        print(f"Error en la línea: {linea.strip()}")
            return self.productos
    except FileNotFoundError:
        print(f"El archivo '{nombre_archivo}' no fue encontrado.")
    except Exception as e:
        print("Ocurrió un error:", e)
```

Función que devuelve una lista nativa con los datos del archivo leído “.inv”, esta función tiene como parámetro el nombre del archivo, este se lee mediante un input y se llama a la función, abre el archivo con el nombre proporcionado en el modo de lectura “r” y con la codificación UTF-8, lee las líneas del archivo y las almacena en la lista líneas.

Luego itera sobre cada línea en la lista “líneas”. Si una línea comienza con “crear_producto ” es importante el espacio al final, se procede a procesarla. Se elimina cualquier espacio en blanco al principio y al final de la línea y luego se divide la línea en partes utilizando el espacio en blanco como separador. La segunda parte, o sea el índice [1] de esta división se divide nuevamente utilizando el punto y coma que es el separador de los archivos para la práctica, esto resulta en una lista llamada “partes”.

Si la longitud de la lista “partes” es igual a 4, entonces se asume que la línea contiene la “información correcta” no verifica que el precio sea un double por ejemplo. A las partes individuales se les asignan variables, nombre, cantidad, precioUni y ubicación, luego se crea una instancia de la clase Product con los valores extraídos y se agrega a la lista productos.

Si la longitud no es igual a 4, se imprime un mensaje de error en la línea en la que fue encontrado.

Luego se realiza un except por si no se encuentra el archivo en el directorio y si ocurre cualquier otro error de lectura o procesamiento del archivo, se realiza el except Exception y se imprime un mensaje indicando que hubo un error.

```
def vender_producto(self, nombre_producto, cantidad, ubicacion):
    producto_encontrado = None

    for producto in self.productos:
        if producto.nombre == nombre_producto and producto.ubicacion == ubicacion:
            producto_encontrado = producto
            break

    if producto_encontrado is None:
        print(f"-No se encontró el producto '{nombre_producto}' en la ubicación {ubicacion}")
        return

    if producto_encontrado.cantidad < cantidad:
        print(f"-No hay suficiente cantidad de '{nombre_producto}' en la ubicación {ubicacion}")
        return

    producto_encontrado.cantidad -= cantidad
    print(f"+Se vendieron {cantidad} unidades de '{nombre_producto}' en la ubicación {ubicacion}")
```

Este método tiene como parámetros, el nombre del producto, la cantidad y la ubicación, se crea una variable `producto_encontrado` inicializada en null, luego se inicia un bucle `for` que recorre todos los productos en la lista `productos`, dentro del bucle se verifica si el nombre del producto y la ubicación coinciden con los parámetros, si se encuentra el producto, se le asigna el valor a la variable “`producto_encontrado`” y se rompe el bucle con un `break`.

Una vez se haya salido del bucle, se verifica si se encontró el producto, si no se encontró se imprime un mensaje de error y sale del método, después se verifica si la cantidad disponible es menor que la cantidad a vender, una de las restricciones en el enunciado, si es así muestra el mensaje de error y se sale del método, si ambas condiciones no se cumplen, se “vende el producto” reduciendo la cantidad existente con la cantidad vendida indicada en la instrucción.

```
def agregar_stock(self, nombre_producto, cantidad, ubicacion):
    producto_encontrado = None

    for producto in self.productos:
        if producto.nombre == nombre_producto and producto.ubicacion == ubicacion:
            producto_encontrado = producto
            break
    if producto_encontrado is None:
        print(f"-No se encontró el producto '{nombre_producto}' en la ubicación '{ubicacion}'. No se puede agregar stock.")
        return
    producto_encontrado.cantidad += cantidad
    print(f"+Se agregaron {cantidad} unidades de '{nombre_producto}' en la ubicación '{ubicacion}'.")
```

Este método tiene los mismos parámetros que el método anterior, realiza la misma lógica del producto_encontrado, pero este contiene una verificación distinta, si el producto no se encuentra en la lista

del inventario o sea que la variable `producto_encontrado` sea igual a `null` una vez haya salido del bucle, mostrará un mensaje de error, si esta condición no se cumple, procede a agregar la cantidad especificada en la instrucción al objeto que se encontró en el bucle y se muestra un mensaje de éxito.

```
def generar_informe(self, nombre_archivo):
    productos_ordenados = sorted(self.productos, key=lambda producto: producto.ubicacion[-1])
    with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
        archivo.write("Informe de Inventario:\n")
        archivo.write("{:<15} {:<10} {:<15} {:<15} {:<10}\n".format("Producto", "Cantidad", "Precio
Unitario", "Valor Total", "Ubicación"))
        archivo.write("-" * 68 + "\n")

        for producto in productos_ordenados:
            valor_total = producto.cantidad * producto.precioUni
            archivo.write("{:<15} {:<10} {:<15.2f} {:<15.2f} {:<10}\n".format(producto.nombre,
producto.cantidad, producto.precioUni, valor_total, producto.ubicacion))


        archivo.write("-" * 68 + "\n")
```

Este método tiene como parámetro el nombre del archivo que se generará: se crea una nueva lista llamada `productos_ordenados` utilizando la función `sorted()`, la lista de productos almacenada en `productos` se ordena en función de la última letra de la ubicación de cada producto. Se utiliza la función `lambda` como clave “key” para especificar cómo se debe realizar la ordenación.

Luego se abre el archivo en modo de escritura "w" y UTF-8 encoding y se le asigna la variable archivo, se escribe una línea en el archivo que dice "Informe de inventario:", se escribe el encabezado con formato y con los campos "Producto", "Cantidad", "Precio Unitario", "Valor total" y "Ubicación".

Se escribe una línea de guiones para separar el encabezado del contenido, como en el ejemplo del enunciado, luego un bucle recorre cada producto en la lista "productos_ordenados", a cada producto se le calcula el valor total multiplicando la cantidad existente por el precio unitario, y se escribe una línea en el archivo con la información del producto encontrado.

Al final del bucle se escribe otra línea de guiones como el ejemplo del enunciado.



```
def leer_instrucciones(self, nombre_archivo):
    #número = 0
    try:
        with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
            lineas = archivo.readlines()

            for linea in lineas:
                if linea.startswith("agregar_stock "):
                    partes = linea.strip().split(" ")[1].split(";")
                    if len(partes) == 3:
                        nombre, cantidad, ubicacion = partes
                        self.agregar_stock(nombre, int(cantidad), ubicacion)

                if linea.startswith("vender_producto "):
                    partes = linea.strip().split(" ")[1].split(";")
                    if len(partes) == 3:
                        nombre, cantidad, ubicacion = partes
                        self.vender_producto(nombre, int(cantidad), ubicacion)

            return True
    except FileNotFoundError:
        print(f"El archivo '{nombre_archivo}' no fue encontrado.")
    except Exception as e:
        print("Ocurrió un error:", e)
    return False
```

Por último se tiene la función que retorna el valor “True” solo para indicar que el proceso fue exitoso.

Esta función realiza lo mismo que la función “leer_archivo”, con el cambio específico en que ahora tiene 2 condicionales, para leer los 2 tipos de instrucciones que vienen en el archivo y cada instrucción se realizará dependiendo de lo que se defina en el archivo, cada condición llama a los métodos correspondientes, ya sea vender o agregar, de nuevo al final 2 except para los errores no contemplados y por si no se encuentra el archivo en el directorio.