Universidad Tecnológica Nacional
Facultad Regional Avellaneda

UTN AVELLANEDA
Tecnicatura Universitaria en Programación

Materia:	Programación 1			
Pertenece a:	Primer Parcial			
Apellido(1):			Fecha:	28 may 2024
Nombre/s(1):			Docente/s(2):	Bustamante Mathias Ferrini Lucas
División(1):			Nota(2):	
DNI(1):			Firma(2):	
Instancia(2)(3):	**F**	**x**	**RF**	

(1) Campos a completar solo por el alumno.
(2) Campos a completar solo por el docente.
(3) Las instancias válidas son: Final (**F**), Recuperatorio de Final (**RF**). Marcar lo que corresponda con una cruz.

Condiciones de aprobación:

[PARTE 1] Lograr resolver estructura, ABM y listados.
[PARTE 2] Lograr resolver gestión con archivos, filtros, búsquedas y ordenamientos.

Enunciado:

Necesitamos armar un sistema de gestión para la compra/venta de productos importados.
El mismo se encargará de gestionar productos a importar, un control de inventario, venta de productos en stock y depósitos para alojar los productos.

[PARTE 1]

(**class Producto**) Desarrollar la clase Producto, que contará con los siguientes atributos.
CODIGO: Identificador único. Formato: NNNN-XX donde N es numérico y X alfabético. Validar formato con expresiones regulares.
DETALLE: Nombre del producto. Mínimo 1 carácter, Maximo 25 carácteres.
USD COMPRA: Valor de importación.
USD VENTA: Valor de comercialización.
PESO: Peso unitario del producto en gramos.

(**Inventario**) Desarrollar dos vectores que se comporten de manera paralela, donde un vector maneja *códigos de productos* y el otro vector, la *cantidad total de productos* (este valor debe coincidir con la sumatoria distribuida en los depósitos utilizado en [PARTE 2]).

Programa [PARTE 1]

1 - ALTA DE PRODUCTO: Desarrollar la función *producto_alta(path: str, inv1: list, inv2: list) -> bool*, donde creará un nuevo producto y lo guardará en nuestra base de datos DB_PRODUCTOS.csv. Validar que no exista en nuestra base de datos el mismo código, de ser válida el alta, agregar el código de producto a nuestro inventario junto con la cantidad inicial en cero y guardarlo en nuestra base de datos. Devolverá un boolean por OK o FAIL. Informar por error: "[ERROR] Ítem existente."
2 - BAJA DE PRODUCTO: Desarrollar la función *producto_baja(path: str, inv1: list, inv2: list) -> bool*, se encargará de listar todos los productos dados de alta, se seleccionara uno y solo se podrá dar de baja si la misma su cantidad en inventario es de cero; de ser asi, dar de baja en nuestra base de datos y en nuestro inventario. Informar por error: "[ERROR] Ítem con stock."
3 - MODIFICAR PRODUCTO: Desarrollar la función *producto_modificar_compra(path: str, codigo: str)* y *producto_modificar_venta(path: str, codigo: str)*, ambas funciones se encargan de setear un nuevo precio para la compra/venta del producto. Validar precio positivo. La modificación afectará a nuestra base de datos.
4 - LISTADO DE PRODUCTOS: Desarrollar la función *producto_listar(path: str, inv1: list, inv2: list)*, donde desde nuestra base de datos e inventario, tome la información necesaria para listar la siguiente información: [CODIGO] [DETALLE] [PRECIO COMPRA] [PRECIO VENTA] [CANTIDAD EN INVENTARIO]

[PARTE 2]

(**class Deposito**) Desarrollar la clase Depósito, que contará con los siguientes atributos.
ID: Identificador único autoincremental. Por cada nuevo depósito, el constructor deberá gestionar el incremento del ID. No se recibe como argumento en constructor.
CAPACIDAD: Unico argumento recibido por constructor. Numérico positivo. Gestiona la capacidad máxima de productos que puede almacenar.
STOCK: Lista de diccionarios. Formato {"ítem": código, "cantidad": stock}.
Valor de ítem: código de producto. Valor de cantidad: stock ingresado al depósito.
LEN: Reemplazar el funcionamiento de la función len() de nuestro objeto para que la misma retorne la capacidad disponible en nuestro depósito.

(**class Gestion**) Desarrollar la clase Gestión, que no tendrá atributos, pero sí funciones.
IMPORTAR(args[]): Los argumentos serán los necesarios para realizar el siguiente comportamiento:

- Se tendrá que listar los productos de nuestra base de datos.
- Elegir el producto junto con la cantidad a importar.
- Guardar nuestros productos importados en nuestro inventario, si ya existe la cantidad, acumularlo.
- Cuando se hace una importación los productos deben ser guardados en un depósito, para esto, validar la capacidad disponible de nuestros depósitos. Si ya existe el producto en nuestro depósito, acumular la cantidad importada. Si la capacidad de almacenamiento de nuestro depósito no alcanza, guardar la capacidad disponible y el resto en un nuevo depósito (ejemplo: deposito_1 tiene la capacidad total de 1000 unidades, está disponible 200 unidades y se quiere hacer un importe de 500 productos. Guardar en deposito_1 las 200 unidades importadas y las 300 unidades restantes guardarlas en un nuevo deposito_2)
- Guardar el estado de nuestros depósitos en un archivo DEPOSITOS.json

VENDER(args[]): Los argumentos serán los necesarios para realizar el siguiente comportamiento:

- Listar los productos disponibles en nuestro inventario que contengan stock superiores a cero.
- Solicitar CUIT del comprador y la cantidad a vender. Formato de CUIT: NN-NNNNNNNN-N, validar con expresiones regulares. La cantidad de compra no puede superar a la disponible en el inventario.
- Cuando se realice la compra se descontará tanto de nuestro inventario como de nuestros depósitos.
- Si se vende todo, quedará nuestro inventario en cero y nuestros depósitos sin este producto.
- Actualizar espacio disponible en depósitos una vez realizada la venta.
- Si nuestro depósito no tiene ningún producto, o sea, este vacío y su capacidad disponible en la máxima. Eliminar el mismo de nuestra lista de depósitos.
- Actualizar nuestra lista de depósitos.
- Generar ticket de venta, para esto, se solicitará la cotización del momento y se la pasará como argumento a la función GENERAR_TICKET(...).

GENERAR_TICKET(cotización: float, args[]): Los argumentos serán los necesarios para realizar el siguiente comportamiento:

- El atributo cotización se solicitará al momento de tener la venta confirmada, se utiliza para realizar la conversión de precio de venta USD del producto a pesos argentinos ARS.
- El ticket escribirá un print en pantalla y un archivo ventas.txt apendeando todas las ventas realizadas.
- El ticket escrito en nuestro archivo de ventas.txt tendrá como cabecera la fecha y hora al momento de la compra, utilizar el siguiente algoritmo:

```
from datetime import datetime
fecha_hora_actual = datetime.now()
formato = fecha_hora_actual.strftime("%d/%m/%Y %H:%M")
print("Fecha y hora actual:", formato) #ej: 23/05/2024 15:30
```

- Tanto el ticket impreso por pantalla como el guardado en ventas.txt tendrá esta información:
  [VENTA dd/mm/yyyy hh:mm]
  CUIT COMPRADOR:
  DETALLE PRODUCTO:
  CANTIDAD VENDIDA:
  IMPORTE USD:
  IMPORTE ARS:
  COTIZACIÓN DEL DIA:

LOG(mensaje: str): Esta función será la encargada de escribir el estado actual de nuestro programa. La implementación será donde crean conveniente.

- Ejemplo de utilización: Altas, bajas, modificación, errores en ejecución/lectura de archivos, etc.
- Se debe lograr escribir con el siguiente formato: *fecha_hora : mensaje*.
  26/05/2024 20:35 : Alta de producto CODIGO 5547-AF
  26/05/2024 20:40 : Venta CUIT 23-55685417-6 importe ARS $157.254
  26/05/2024 20:35 : [ERROR] Alta de CODIGO 5547-AF existente

Programa [PARTE 2]

6 - IMPORTAR PRODUCTO: Utilizar la función importar(...) de la clase Gestion.
7 - VENDER PRODUCTO: Utilizar la función vender(...) de la clase Gestion.
8 - LISTAR ESTADO DE DEPOSITOS: Desarrollar la función *deposito_listar(...)*, donde listara el estado actual de cada depósito con productos, informando:
[ID de depósito] - [CAPACIDAD ocupada/máxima] - [DISPONIBLE]ej:
ID 1000 - 150/1000 - Disponible 850
ID 1001 - 250/900 - Disponible 650
9 - BUSCAR PRODUCTO CON MENOS STOCK: Informar el producto con menos stock y listar en los depósitos en los que se encuentre. EJ:
[PRODUCTO 5547-AF]
TOTAL STOCK: 150
Depósito 1: 75
Depósito 3: 25
Depósito 5: 50
10 - FILTRAR PRODUCTOS POR PRECIO DE MEDIA: Listar todos los productos cuyo precio de compra sea inferior al precio medio de venta.
11 - ORDENAR DEPÓSITOS POR PESO: Ordenar y listar todos depósitos de manera ascendente, cuyo criterio sea por peso total. Para esto, utilizar el peso de cada producto.