Manual Técnico

Enrique Fernando Gaitán Ibarra

202110531

Contenido

Información General 3
Paradigmas 3
Diagrama de clase 3
Módulos y Archivos 4
db 4
Carpeta 4
Archivo – database 4
helpers 4
Carpeta 4
Archivo - Inquirer 5
Archivo - toOutFile 7
Archivo - uploadStock 8
Archivo - utils 9
models 10
Carpeta 10
Archivo - Product 10
Paquetes 11

Información General

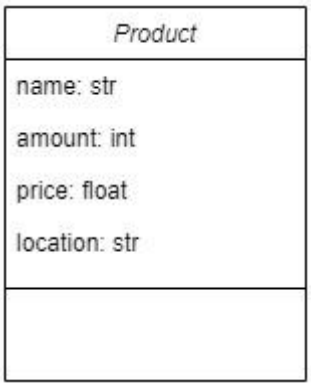
- Práctica: Gestor de Inventario
- Desarrollador: Enrique Fernando Gaitán Ibarra
- Lenguaje: Python
- Tipo: Aplicación de consola

Paradigmas

Se utiliza la programación orientada a objetos para realizar un modelo de productos, y una separación de archivos según su funcionalidad en el programa, por lo que, puede ser indicar que esta organizado de forma modular

Diagrama de clase

Para poder tener un molde de la información a almacenar, se realizó una clase llamada *Product*, en la cual se describen los atributos que se desea utilizar durante todo el programa.

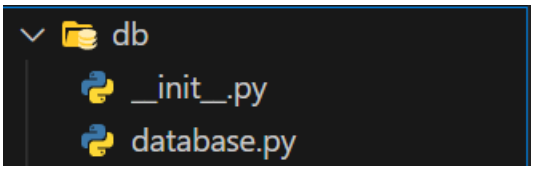


Módulos y Archivos

db

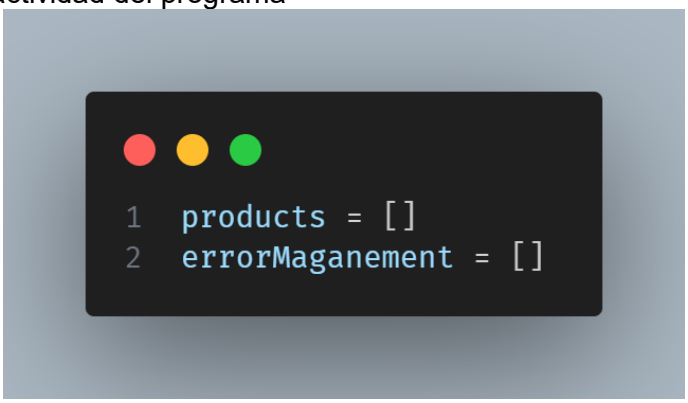
Carpeta

- Descripción: En este módulo se encuentra las listas -list-, que se utilizan a lo largo del programa, como similitud de una tabla en una base de datos para guardar la información.



Archivo – database

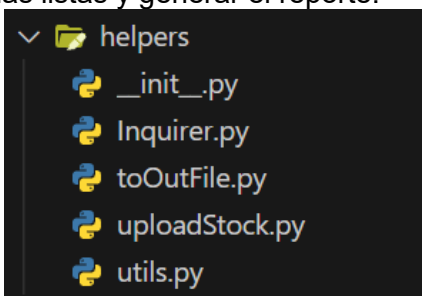
- products: Lista donde se almacena a través de diccionarios -dict- individuales, la información de cada producto.
- errorMaganement: Lista en la cual se almacenan los errores que se obtengan durante la actividad del programa



helpers

Carpeta

- Descripción: En este módulo se ubica el menú principal, los métodos leer y escribir archivos, guardar en las listas y generar el reporte.



Archivo - Inquirer

Métodos o Funciones

- validMenu: función que valida la opción seleccionada por el usuario en el menú principal, y lo redirige a donde este desea.

```python
def validMenu( optionSelected: str ):
    if ( optionSelected == "Cargar inventario inicial" ):
        helpers.onStartChangeDB()
    elif ( optionSelected == "Cargar instrucciones de movimientos" ):
        helpers.onStartChangeDB()
    elif ( optionSelected == "Crear informe de inventario" ):
        helpers.outTxtFIle(filename="stock.txt", products=products)
    else:
        sys.exit()
```

- pause: Permite realizar una “pausa” en el programa, para que el usuario confirme que los datos o la información ingresada este bien. Se utilizo el paquete *inquirer* para el manejo de respuestas de parte del usuario y tener un mejor aspecto visual.

```python
def pause():
    questions = [
        inquirer.Confirm("continue", message="¿Deseas continuar?"),
    ]

    answers = inquirer.prompt(questions)
    optionSelected = answers["continue"]
    print(optionSelected)
```

- initialMenu: Este es el menú inicial de la aplicación. De igual forma se utilizó el paquete inquirer. Esta función envía el valor que es recibido en validMenu, anteriormente explicado.

```python
def initialMenu():
    os.system('cls')

    menuOptions = [
        inquirer.List(
            name="menu",
            message="¿Qué desea hacer?",
            choices=options,
            carousel=True
        )
    ]

    answers: list = inquirer.prompt(menuOptions)
    optionSelected: int = answers["menu"]
    validMenu( optionSelected )
```

- options: Lista en la cual están almacenadas las opciones que se muestran en el menú principal.

```python
options = [
    "Cargar inventario inicial",
    "Cargar instrucciones de movimientos",
    "Crear informe de inventario",
    "Salir"
]
```

Archivo - toOutFile

Métodos o funciones

- outTxtFile: Función que recibe como parámetros el nombre del archivo a generar y la lista de productos almacenada. Las acciones de la función son los siguientes:
    1. Como primer paso se procede declarar la ruta de salida – path – y encontrar la fecha del día de hoy.
    2. Se utiliza una *lambda comprehension* con *for* e *if*, para omitir los elementos en donde su cantidad actual sea menor a 0 y no tenerlos en la nueva lista.
    3. Se utiliza el método *sort* nativo de Python para ordenar la lista, usando como "clave" el campo *location*.
    4. Se recorre la lista de los productos para ingresarlos a otra, donde, se cambian todos a datos tipo *string*. Con el fin de ser mostrados en el reporte final de esta forma.
    5. Se realiza la tabulación de los datos, usando el paquete *tabulate*.
    6. Con el método *with* y *open*, se crea el archivo y se escribe la información en él.
    7. Se cierra el método with.

```python
def outTxtFIle(filename: str, products = []):
    path = os.path.join(os.getcwd(), filename)
    now = datetime.datetime.now()
    producWithStock = [product for product in products if product.amount > 0 ]
    producWithStock.sort(key=lambda product: product.location)
    productTablePrint = [[
                        "Producto",
                        "Cantidad",
                        "Precio Unitario",
                        "Valor Total",
                        "Ubicación"]
                        ]
    for product in producWithStock:
        productRow = [
            product.name, product.amount,
            f'Q { product.price }',
            f'Q { product.price * product.amount }',
            product.location
        ]
        productTablePrint.append(productRow)
    tabulate(productTablePrint,headers="firstrow")
    with open(path, "w+", encoding="utf-8") as f:
        f.write('                        Informe de Inventario:\n')
        f.write(tabulate(productTablePrint,
                    headers="firstrow",
                    showindex="always",
                    tablefmt="simple_grid",
                    numalign="center")
                )
        f.write('\n')
        f.write(F'  Generado el { now.strftime("%m-%d-%Y %H:%M:%S") }')
        f.close()
    helpers.pause()
    helpers.initialMenu()
```

Archivo - uploadStock

Métodos o funciones

- onStartChangeDB: Función que recibe cada línea leída en el archivo ingresado con el objetivo de repararla según y redirigirla a su método respectivo, según la información obtenida.

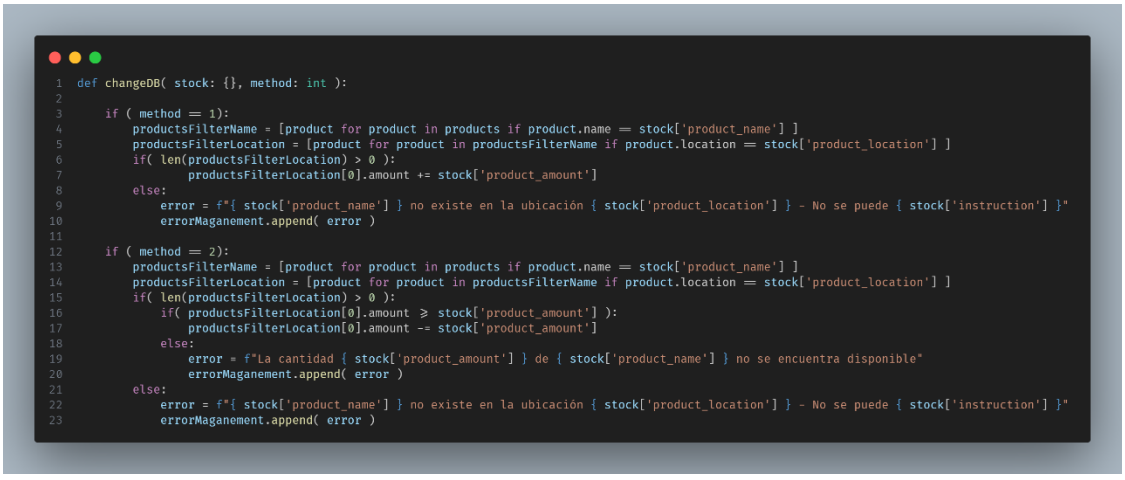
```python
def onStartChangeDB() -> None:
    stocks = helpers.read()
    for stock in stocks:
        if stock['instruction'] == helpers.optionInstruction[0]:
            chargeDB( stock )
        elif stock['instruction'] == helpers.optionInstruction[1]:
            changeDB( stock, 1 )
        else:
            changeDB( stock, 2 )
    else:
        stocks.clear()

    if( len(errorMaganement) > 0 ):
        for i, error in enumerate(errorMaganement):
            print(f"Error { i + 1 }: { error }")
    helpers.pause()
    helpers.initialMenu()
```

- chargeDB: En esta parte se crea el objeto de tipo Product con la información recibida en el parámetro y lo agrega a la lista de productos.

```python
def chargeDB( stock: {} ):
    product: Product = Product(name=stock['product_name'], amount=stock['product_amount'], price=stock['product_price'], location=stock['product_location'])
    products.append(product)
```

- changeDB: Esta función es la encargada de válida las acciones obtenidas del archivo. Se separa en 2 métodos, uno para agregar (1) y otro para eliminar (2) productos. En ambos casos se utilizaron *lambdas comprehension,* para la clasificación de la información y obtención de errores (si fuera el caso).



```python
def changeDB( stock: {}, method: int ):

    if ( method == 1):
        productsFilterName = [product for product in products if product.name == stock['product_name'] ]
        productsFilterLocation = [product for product in productsFilterName if product.location == stock['product_location'] ]
        if( len(productsFilterLocation) > 0 ):
                productsFilterLocation[0].amount += stock['product_amount']
        else:
            error = f"{ stock['product_name'] } no existe en la ubicación { stock['product_location'] } - No se puede { stock['instruction'] }"
            errorMaganement.append( error )

    if ( method == 2):
        productsFilterName = [product for product in products if product.name == stock['product_name'] ]
        productsFilterLocation = [product for product in productsFilterName if product.location == stock['product_location'] ]
        if( len(productsFilterLocation) > 0 ):
            if( productsFilterLocation[0].amount >= stock['product_amount'] ):
                productsFilterLocation[0].amount -= stock['product_amount']
            else:
                error = f"La cantidad { stock['product_amount'] } de { stock['product_name'] } no se encuentra disponible"
                errorMaganement.append( error )
        else:
            error = f"{ stock['product_name'] } no existe en la ubicación { stock['product_location'] } - No se puede { stock['instruction'] }"
            errorMaganement.append( error )
```

Archivo - utils

Métodos o funciones

- read: Función encargada de “limpiar” cada una de las líneas del archivo y clasificar estas según el comando inicial (instrucción) recibido.

```python
def read() -> []:
    stocksGroup = []
    path: str = pathRequired('Ingresa la ruta del archivo')
    file1 = open(path, mode = 'r', encoding='utf-8')
    count = 0

    while True:
        count += 1

        line = file1.readline()

        # Cleanning lines
        auxline = line.strip()

        if not auxline:
            break

        arrayWords = auxline.split()
        instruction = arrayWords[0]
        product = arrayWords[1].split(";")

        if ( instruction == optionInstruction[0]):
            product_dict = {
                "instruction":instruction,
                "product_name": product[0],
                "product_amount": int(product[1]),
                "product_price": float(product[2]),
                "product_location": product[3],
            }

        if ( instruction == optionInstruction[1]):
            product_dict = {
                "instruction":instruction,
                "product_name": product[0],
                "product_amount": int(product[1]),
                "product_location": product[2],
            }

        if ( instruction == optionInstruction[2]):
            product_dict = {
                "instruction":instruction,
                "product_name": product[0],
                "product_amount": int(product[1]),
                "product_location": product[2],
            }

        stocksGroup.append(product_dict)
    file1.close()
    return stocksGroup
```

- Path: Únicamente se encarga de leer la entrada del archivo y validar que esta sea una ruta que apunte a un archivo

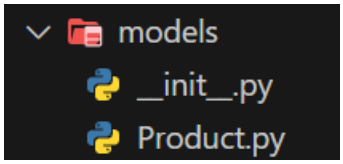
```python
def pathRequired( mess ) -> str:
    questions = [
    inquirer.Path(
        name='path',
        message=mess,
        path_type=inquirer.Path.FILE,
        ),
    ]

    answers: dict = inquirer.prompt(questions)
    path: str = answers["path"]
    return path
```

models

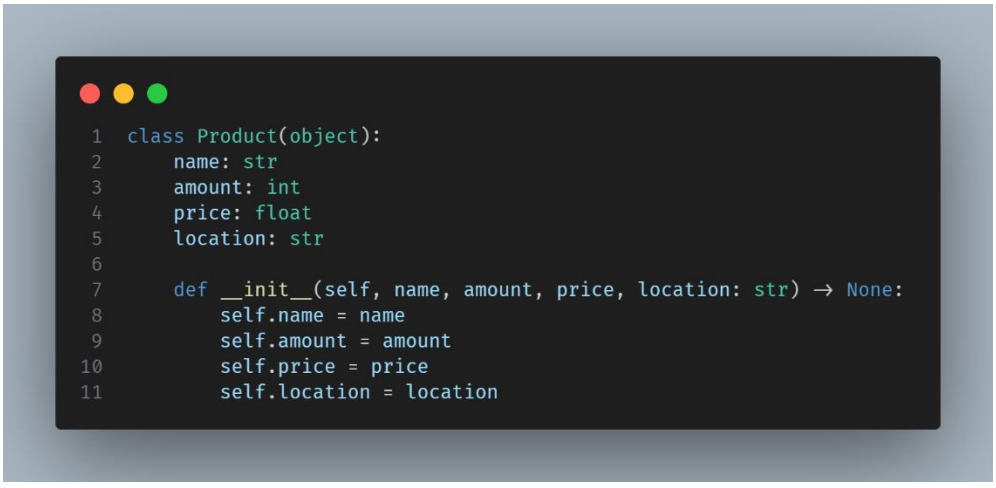
Carpeta

- Descripción: En este módulo se encuentra el modelo de la clase Producto.



Archivo - Product

- Product: Con únicamente su constructor, la clase Product, recibe 4 parámetros el nombre – name- y la ubicación – location- de tipo string, cantidad – amount - de tipo entero y precio – price – de tipo float.



```python
class Product(object):
    name: str
    amount: int
    price: float
    location: str

    def __init__(self, name, amount, price, location: str) -> None:
        self.name = name
        self.amount = amount
        self.price = price
        self.location = location
```

Paquetes

- Inquirer
- Tabulate