Republic of the Philippines
SURIGAO DEL NORTE STATE UNIVERSITY
Narciso Street, Surigao City 8400, Philippines

"For Nation's Greater

In Partial Fulfilment of the Requirement for the course of

CS 223 - Object-Oriented Programming

SIMPLIFIED INVENTORY MANAGEMENT SYSTEM FOR TIKTOKSHOP

Presented to:

Dr. Unife O. Cagas

Professor

Prepared by:

Jumar C. Guardalupe

BSCS-2A: Computer Science

May. 2024

SIMPLIFIED INVENTORY MANAGEMENT SYSTEM FOR TIKTOKSHOP

The TikTok Shop project is a Python-based program designed to simulate a digital store environment where users can buy various items such as outfits and gadgets. Inspired by the popular social media platform, TikTok, the project aims to create a fun and interactive shopping experience for users. The TikTok Shop project serves as a practical example of object-oriented programming in Python, demonstrating concepts such as inheritance, encapsulation, and polymorphism. It provides a foundation for building more complex e-commerce systems and can be expanded with additional features such as user authentication, shopping cart functionality, and payment processing.

OBJECTIVES:

- To implement encapsulation using underscores to protect attributes.
- To demonstrate inheritance by creating subclasses that inherit from a parent class.
- To showcase polymorphic behavior by overriding methods in subclasses.
- To enable a functional selling system considering stock availability.

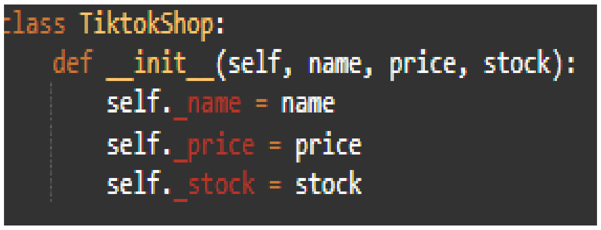
IMPORTANCE & CONTRIBUTION OF THE PROJECT:

The TikTok Shop project presents a practical application of object-oriented programming (OOP) principles, serving as a valuable educational tool for learners. Through the implementation of classes and inheritance, it illustrates core OOP concepts such as encapsulation, inheritance, and polymorphism. By organizing code into modular components, the project promotes code reusability and modularity, facilitating easier maintenance and extension of the system. This real-world simulation of an online shop provides learners with hands-on experience in solving problems related to inventory management, product selling, and user interaction, thereby enhancing their problem-solving skills and understanding of software development principles.

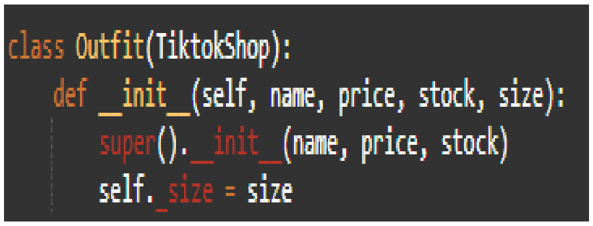
Moreover, the TikTok Shop project serves as a foundational resource for further learning and development. Learners can build upon the project to add more features and complexity, such as user authentication, shopping cart functionality, and database integration. This progression allows individuals to explore advanced topics in software development while reinforcing their understanding of Python programming and OOP concepts. Overall, the project not only equips learners with practical skills applicable to software development but also fosters a deeper comprehension of programming principles through experiential learning.

PRINCIPLES OF OOP BEING USED IN THE CODE:

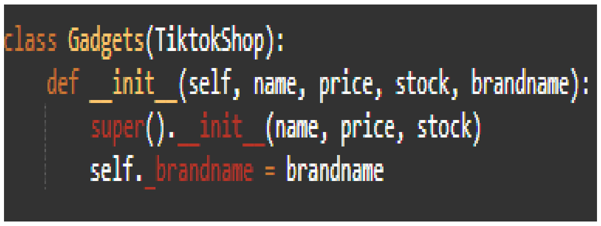
Inheritance is utilized in the code to establish a hierarchical relationship between classes, enabling the creation of specialized classes like Outfit and Gadgets that inherit attributes and methods from the more general TiktokShop class. This hierarchical structure promotes code reusability and facilitates maintenance by allowing common functionality to be defined in one place and reused across multiple subclasses. By leveraging inheritance, developers can efficiently organize and extend the codebase, reducing redundancy and enhancing modularity.



```
class TiktokShop:
    def __init__(self, name, price, stock):
        self._name = name
        self._price = price
        self._stock = stock
```

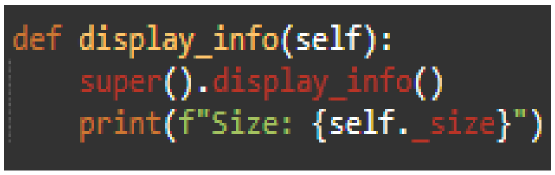


```
class Outfit(TiktokShop):
    def __init__(self, name, price, stock, size):
        super().__init__(name, price, stock)
        self._size = size
```

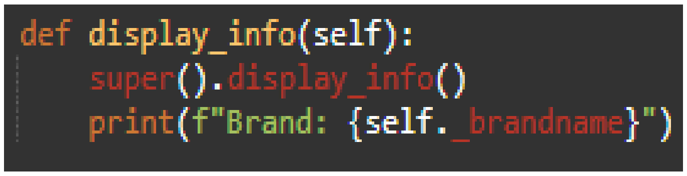


```
class Gadgets(TiktokShop):
    def __init__(self, name, price, stock, brandname):
        super().__init__(name, price, stock)
        self._brandname = brandname
```

Polymorphism is demonstrated in the display_info() method, where each subclass implements its own version to provide specific information about the item being sold. This enables objects of different types to be treated uniformly through a common interface, promoting flexibility and extensibility in the code. Polymorphism simplifies code maintenance and promotes scalability by allowing new subclasses to be added without modifying existing code, thereby supporting the open-closed principle.

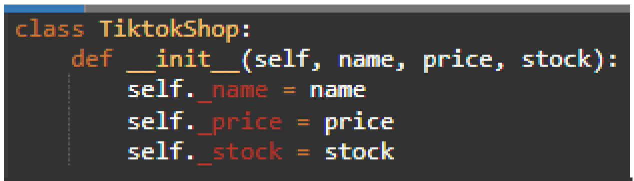


```
def display_info(self):
    super().display_info()
    print(f"Size: {self._size}")
```

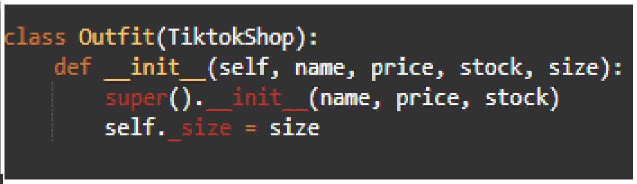


```
def display_info(self):
    super().display_info()
    print(f"Brand: {self._brandname}")
```

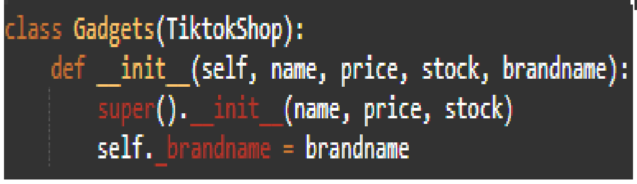
Encapsulation is practiced by encapsulating data within classes and controlling access to it through methods like getters and setters. In the code, encapsulation is indicated by using underscores to denote protected attributes, which ensures data integrity and restricts direct access to class variables from outside the class. Encapsulation enhances code maintainability and promotes modular design by isolating implementation details within classes, reducing dependencies and making the codebase more resilient to changes.



```
class TiktokShop:
    def __init__(self, name, price, stock):
        self._name = name
        self._price = price
        self._stock = stock
```

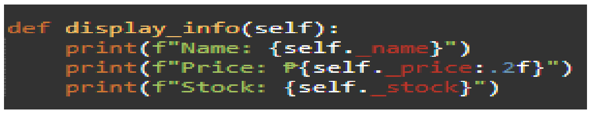


```
class Outfit(TiktokShop):
    def __init__(self, name, price, stock, size):
        super().__init__(name, price, stock)
        self._size = size
```

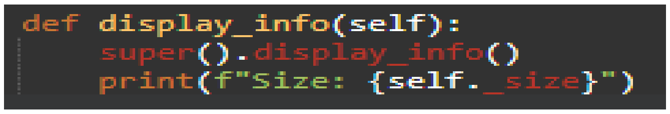


```
class Gadgets(TiktokShop):
    def __init__(self, name, price, stock, brandname):
        super().__init__(name, price, stock)
        self._brandname = brandname
```

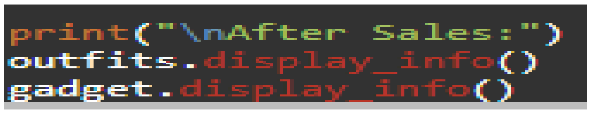
Abstraction is present in the code as it hides internal implementation details and exposes only essential features through a clear and simplified interface. By abstracting away unnecessary complexity, abstraction helps in managing the complexity of the codebase and promotes code reuse and modularity. Abstraction allows developers to focus on the essential aspects of the problem domain, making the code easier to understand, maintain, and extend over time.



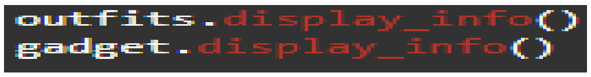
```
def display_info(self):
    print(f"Name: {self._name}")
    print(f"Price: ₱{self._price:.2f}")
    print(f"Stock: {self._stock}")
```



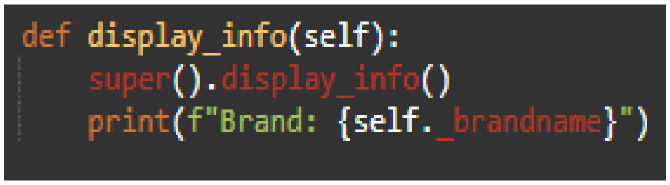
```
def display_info(self):
    super().display_info()
    print(f"Size: {self._size}")
```



```
print("\nAfter Sales:")
outfits.display_info()
gadget.display_info()
```



```
outfits.display_info()
gadget.display_info()
```



```
def display_info(self):
    super().display_info()
    print(f"Brand: {self._brandname}")
```

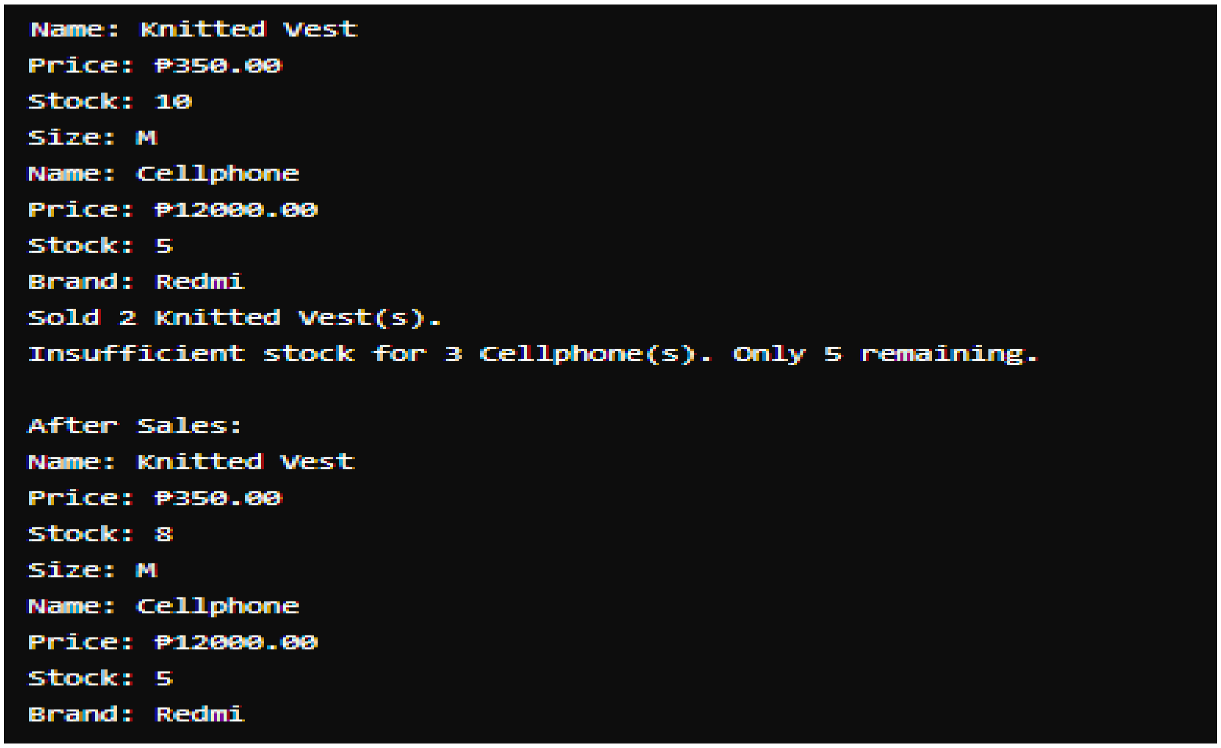
HARDWARE & SOFTWARE USED:

Compiler: Online GDB

Programming Language: Python

Hardware: Laptop

OUTPUT:



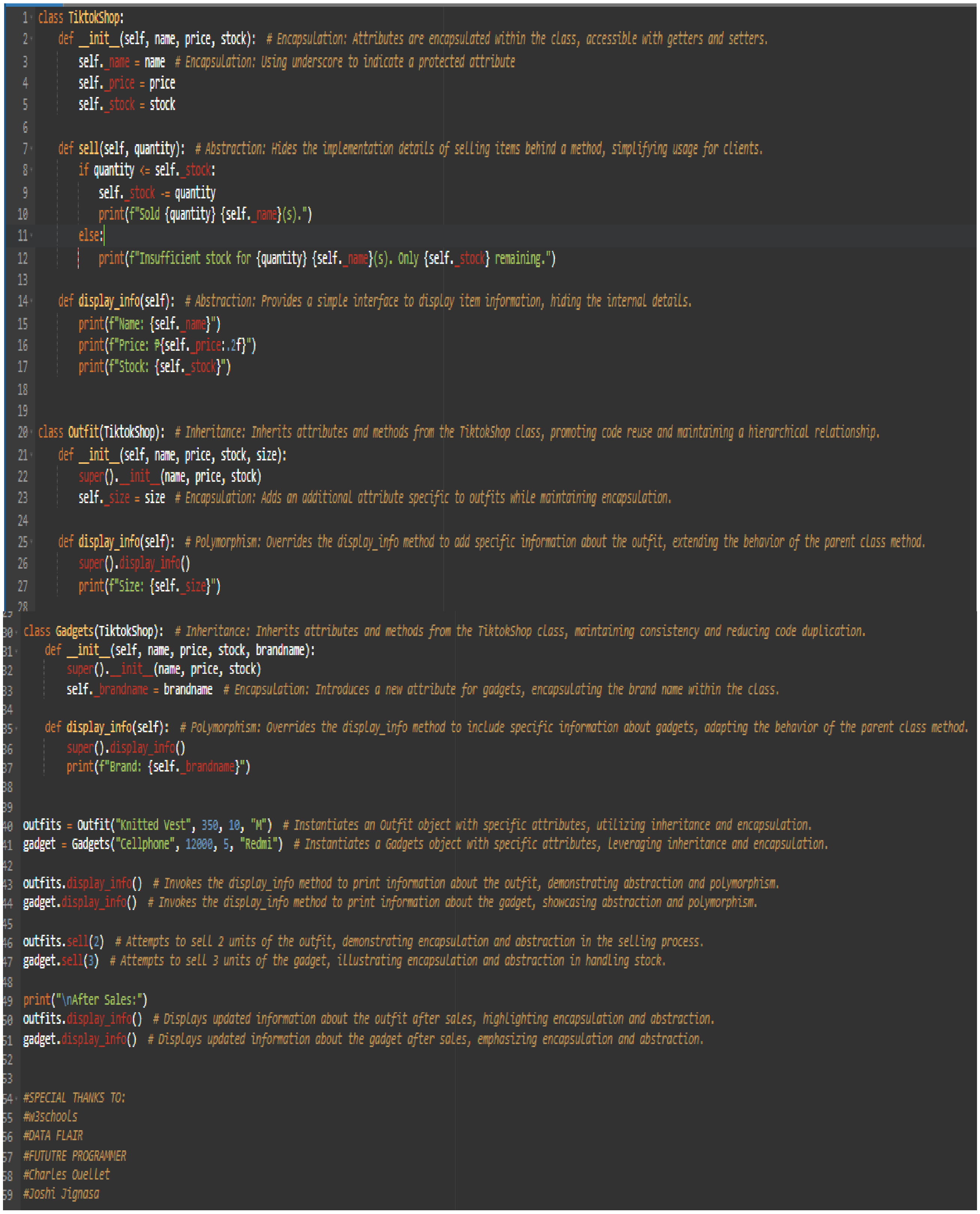
```
Name: Knitted Vest
Price: ₱350.00
Stock: 10
Size: M
Name: Cellphone
Price: ₱12000.00
Stock: 5
Brand: Redmi
Sold 2 Knitted Vest(s).
Insufficient stock for 3 Cellphone(s). Only 5 remaining.

After Sales:
Name: Knitted Vest
Price: ₱350.00
Stock: 8
Size: M
Name: Cellphone
Price: ₱12000.00
Stock: 5
Brand: Redmi
```

OUTPUT DESCRIPTION:

Displaying item information utilizes overridden methods to include unique details like size for outfits and brand for gadgets. Sales attempts decrease stock accordingly, with feedback indicating successful transactions or stock shortages. The system effectively manages inventory and provides clear insights into available stock and item details for efficient business operations.

CODE:



```python
class TiktokShop:
    def __init__(self, name, price, stock):  # Encapsulation: Attributes are encapsulated within the class, accessible with getters and setters.
        self._name = name  # Encapsulation: Using underscore to indicate a protected attribute
        self._price = price
        self._stock = stock

    def sell(self, quantity):  # Abstraction: Hides the implementation details of selling items behind a method, simplifying usage for clients.
        if quantity <= self._stock:
            self._stock -= quantity
            print(f"Sold {quantity} {self._name}(s).")
        else:
            print(f"Insufficient stock for {quantity} {self._name}(s). Only {self._stock} remaining.")

    def display_info(self):  # Abstraction: Provides a simple interface to display item information, hiding the internal details.
        print(f"Name: {self._name}")
        print(f"Price: ₱{self._price:.2f}")
        print(f"Stock: {self._stock}")


class Outfit(TiktokShop):  # Inheritance: Inherits attributes and methods from the TiktokShop class, promoting code reuse and maintaining a hierarchical relationship.
    def __init__(self, name, price, stock, size):
        super().__init__(name, price, stock)
        self._size = size  # Encapsulation: Adds an additional attribute specific to outfits while maintaining encapsulation.

    def display_info(self):  # Polymorphism: Overrides the display_info method to add specific information about the outfit, extending the behavior of the parent class method.
        super().display_info()
        print(f"Size: {self._size}")


class Gadgets(TiktokShop):  # Inheritance: Inherits attributes and methods from the TiktokShop class, maintaining consistency and reducing code duplication.
    def __init__(self, name, price, stock, brandname):
        super().__init__(name, price, stock)
        self._brandname = brandname  # Encapsulation: Introduces a new attribute for gadgets, encapsulating the brand name within the class.

    def display_info(self):  # Polymorphism: Overrides the display_info method to include specific information about gadgets, adapting the behavior of the parent class method.
        super().display_info()
        print(f"Brand: {self._brandname}")


outfits = Outfit("Knitted Vest", 350, 10, "M")  # Instantiates an Outfit object with specific attributes, utilizing inheritance and encapsulation.
gadget = Gadgets("Cellphone", 12000, 5, "Redmi")  # Instantiates a Gadgets object with specific attributes, leveraging inheritance and encapsulation.

outfits.display_info()  # Invokes the display_info method to print information about the outfit, demonstrating abstraction and polymorphism.
gadget.display_info()  # Invokes the display_info method to print information about the gadget, showcasing abstraction and polymorphism.

outfits.sell(2)  # Attempts to sell 2 units of the outfit, demonstrating encapsulation and abstraction in the selling process.
gadget.sell(3)  # Attempts to sell 3 units of the gadget, illustrating encapsulation and abstraction in handling stock.

print("\nAfter Sales:")
outfits.display_info()  # Displays updated information about the outfit after sales, highlighting encapsulation and abstraction.
gadget.display_info()  # Displays updated information about the gadget after sales, emphasizing encapsulation and abstraction.


#SPECIAL THANKS TO:
#W3schools
#DATA FLAIR
#FUTUTRE PROGRAMMER
#Charles Ouellet
#Joshi Jignasa
```

USER GUIDE:

1. Setting Up the Shop:

- Before using the code, ensure you have Python installed on your computer.
- Copy the provided code into a Python file (e.g., tiktok_shop.py).

2. Creating Items:

- To add items to your TikTok shop, you'll primarily work with two classes: Outfit and Gadgets.
- Use the Outfit class to represent outfits and the Gadgets class to represent gadgets.
- When creating an outfit or a gadget, provide the necessary information such as name, price, initial stock, and additional attributes like size for outfits or brand for gadgets.

3. Displaying Item Information:

- After creating instances of outfits and gadgets, you can display their information using the display_info() method.
- This method prints details like name, price, stock, and any additional attributes specific to the item type.

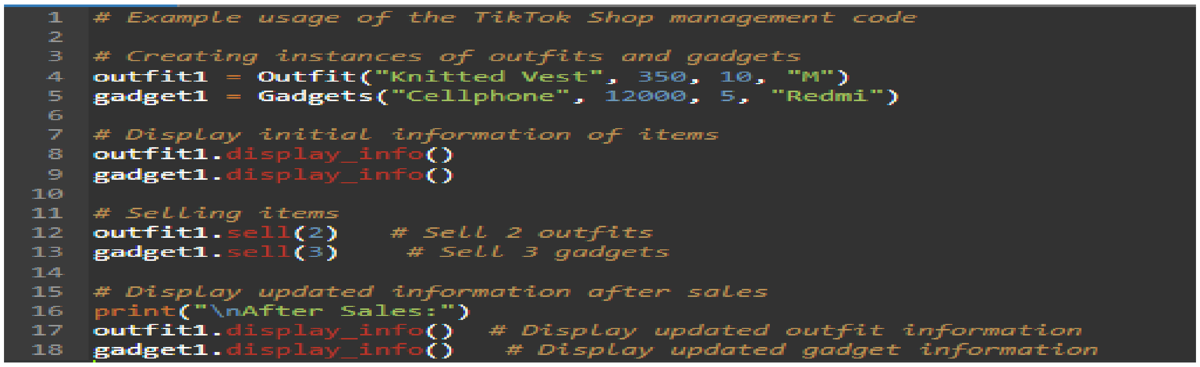
4. Selling Items:

- To sell items from your shop, use the sell(quantity) method on the respective item instance.
- Specify the quantity of items you want to sell. If the quantity is available in stock, it will be sold, reducing the stock count accordingly.
- If the requested quantity exceeds the available stock, a message indicating insufficient stock will be displayed.

5. After Sales:

- After selling items, you can display the updated information about the items using the display_info() method.
- This will show the current stock levels of the items, reflecting the sales made.

Example Usage:



```
# Example usage of the TikTok Shop management code

# Creating instances of outfits and gadgets
outfit1 = Outfit("Knitted Vest", 350, 10, "M")
gadget1 = Gadgets("Cellphone", 12000, 5, "Redmi")

# Display initial information of items
outfit1.display_info()
gadget1.display_info()

# Selling items
outfit1.sell(2)    # Sell 2 outfits
gadget1.sell(3)     # Sell 3 gadgets

# Display updated information after sales
print("\nAfter Sales:")
outfit1.display_info()  # Display updated outfit information
gadget1.display_info()   # Display updated gadget information
```

REFERENCES:

https://medium.com/@joshijignasa67/building-an-online-shopping-cart-with-python-a-dive-into-virtual-grocery-shopping-492766d45f15

https://data-flair.training/blogs/python-online-shopping-system/

https://github.com/topics/online-shop?l=python

https://medium.com/free-code-camp/how-to-build-an-e-commerce-shop-with-python-django-wagtail-3dd2043f89e7

https://www.youtube.com/watch?v=6kCZSYYpV9I