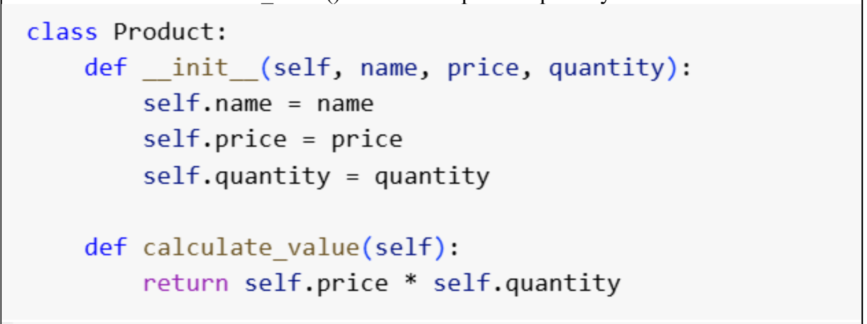
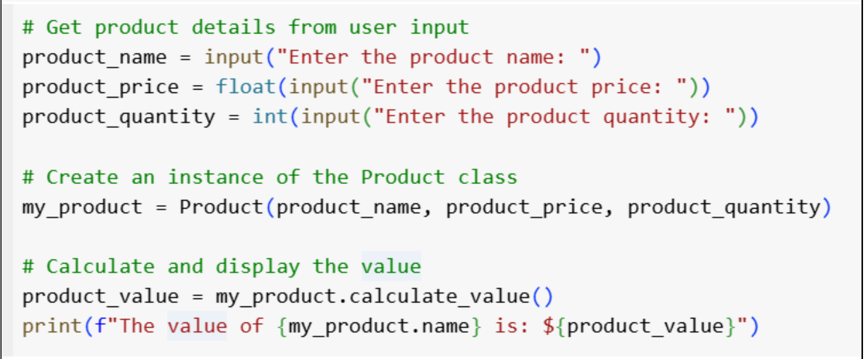
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s)Name		1. Dr. Mohammed Ali Shaik 2. Dr. T Sampath Kumar 3. Mr. S Naresh Kumar 4. Dr. V. Rajesh 5. Dr. Brij Kishore 6. Dr Pramoda Patro 7. Dr. Venkataramana 8. Dr. Ravi Chander 9. Dr. Jagjeeth Singh	
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	06-08-2025	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:6.5(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	**Lab 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals** Lab Assignment 1: Intelligent Code Completion for Object-Oriented Programming **Objective:** To explore AI-powered code assistants for writing Python classes, constructors, and methods through intelligent suggestions. Suppose that you are hired as an intern at a tech company that develops inventory management systems. Your manager asks you to create a **Product** class and a **Warehouse** class with some basic methods. You have decided to use AI-powered code suggestions to help speed up development and reduce syntax errors. Tasks to be completed are as below **1. Setup AI Coding Tool:** • Install and configure GitHub Copilot or Kite with VS Code or JetBrains IDE. • Enable real-time code suggestions. **2. Class Design Using AI Assistance:** • Begin defining a Product class with attributes: name, price, quantity. • Use the AI suggestion feature to automatically complete the __init__() method. • Add a method calculate_value() to return price * quantity. • **Prompt:** Write a Python class named Product with attributes name, price, and quantity. Use a constructor (__init__) to initialize these values, and add a	15.08.2025 EOD

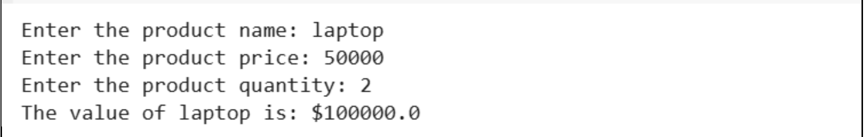
method calculate_value() that returns price * quantity.



```
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_value(self):
        return self.price * self.quantity
```



```
# Get product details from user input
product_name = input("Enter the product name: ")
product_price = float(input("Enter the product price: "))
product_quantity = int(input("Enter the product quantity: "))

# Create an instance of the Product class
my_product = Product(product_name, product_price, product_quantity)

# Calculate and display the value
product_value = my_product.calculate_value()
print(f"The value of {my_product.name} is: ${product_value}")
```

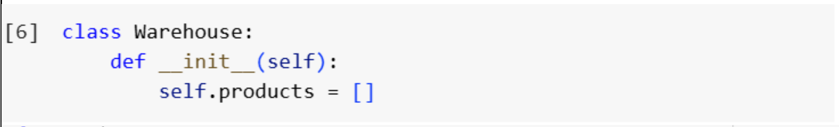


```
Enter the product name: laptop
Enter the product price: 50000
Enter the product quantity: 2
The value of laptop is: $100000.0
```

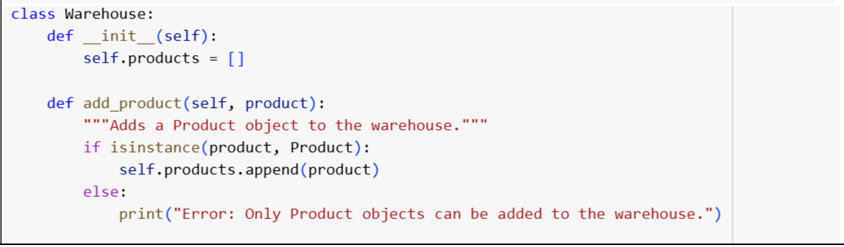
3. Create Another Class:

- Define a Warehouse class with a list of Product objects.
- Use code completion to help implement**:**
    - A method to add a product.
    - A method to display the most valuable product.

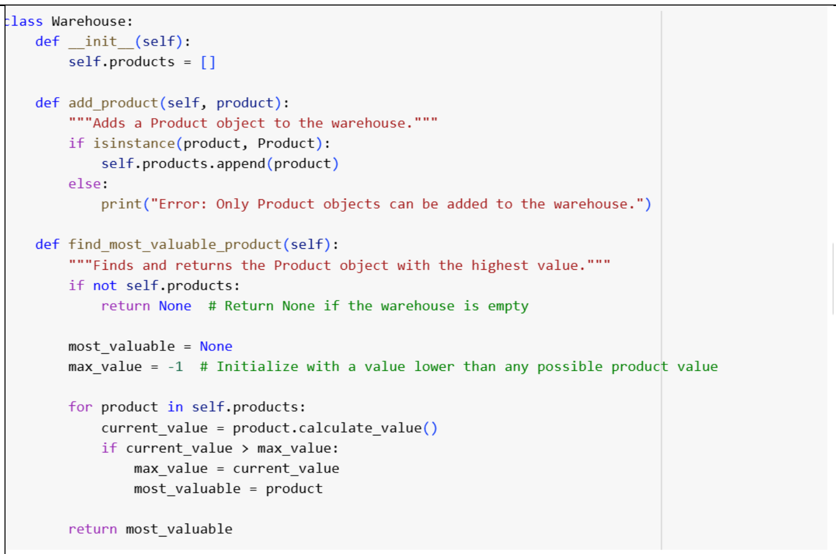
Prompt: Write a Python program with a Warehouse class that stores Product objects. The Warehouse should have methods to add a product and to find the product with the highest total value (price * quantity). Ask the user to enter details for at least three products, add them to the Warehouse, and then print the most valuable product's name and value.



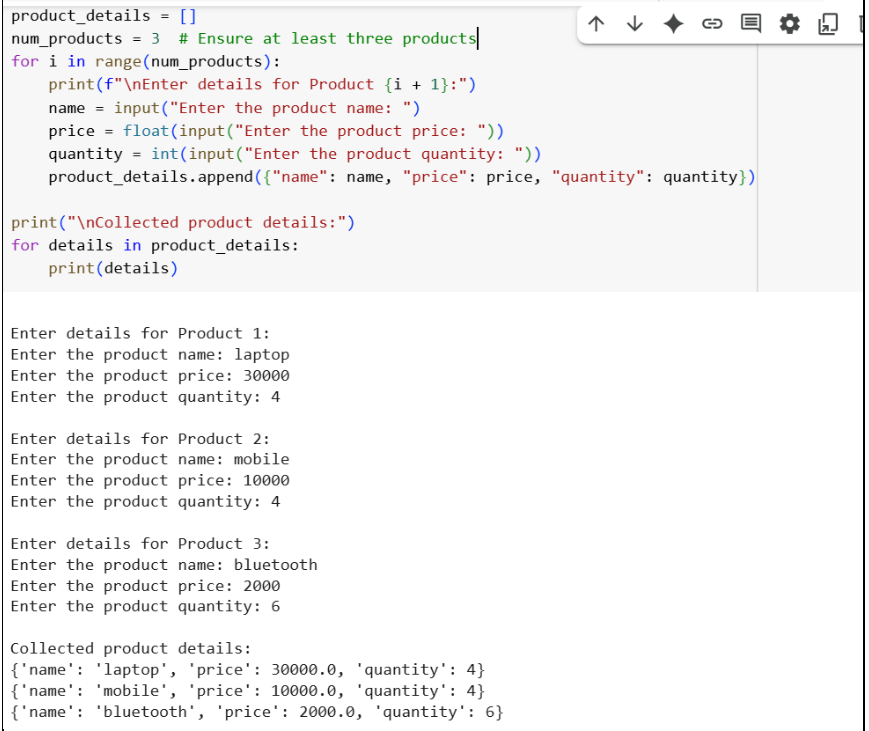
```
[6]  class Warehouse:
         def __init__(self):
             self.products = []
```



```
class Warehouse:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        """Adds a Product object to the warehouse."""
        if isinstance(product, Product):
            self.products.append(product)
        else:
            print("Error: Only Product objects can be added to the warehouse.")
```



```python
class Warehouse:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        """Adds a Product object to the warehouse."""
        if isinstance(product, Product):
            self.products.append(product)
        else:
            print("Error: Only Product objects can be added to the warehouse.")

    def find_most_valuable_product(self):
        """Finds and returns the Product object with the highest value."""
        if not self.products:
            return None  # Return None if the warehouse is empty

        most_valuable = None
        max_value = -1  # Initialize with a value lower than any possible product value

        for product in self.products:
            current_value = product.calculate_value()
            if current_value > max_value:
                max_value = current_value
                most_valuable = product

        return most_valuable
```



```python
product_details = []
num_products = 3  # Ensure at least three products
for i in range(num_products):
    print(f"\nEnter details for Product {i + 1}:")
    name = input("Enter the product name: ")
    price = float(input("Enter the product price: "))
    quantity = int(input("Enter the product quantity: "))
    product_details.append({"name": name, "price": price, "quantity": quantity})

print("\nCollected product details:")
for details in product_details:
    print(details)
```

```

Enter details for Product 1:
Enter the product name: laptop
Enter the product price: 30000
Enter the product quantity: 4

Enter details for Product 2:
Enter the product name: mobile
Enter the product price: 10000
Enter the product quantity: 4

Enter details for Product 3:
Enter the product name: bluetooth
Enter the product price: 2000
Enter the product quantity: 6

Collected product details:
{'name': 'laptop', 'price': 30000.0, 'quantity': 4}
{'name': 'mobile', 'price': 10000.0, 'quantity': 4}
{'name': 'bluetooth', 'price': 2000.0, 'quantity': 6}
```

4. Reflection:

- Identify how much of the code was completed by AI and what manual edits were needed.
- Comment on the relevance and accuracy of AI suggestions.

Product class:

- AI quickly suggested the __init__() constructor with attributes.
- Suggested the calculate_value() method correctly (price * quantity).
- Very little manual editing was required.

Warehouse class:

- AI suggested the __init__() method with a product list.
- Suggested correct code for add_product() using append().
- For most_valuable_product(), AI suggested using max() with a lambda, but I edited it to return the product itself.

	Testing part: • AI suggested hardcoded product objects. • I changed it to take **user input** (name, price, quantity). • This showed AI often provides generic solutions that need human adjustment. **Overall experience:** • About **70% AI-generated, 30% manually edited**. • Suggestions were mostly accurate and helpful. • AI sped up coding and reduced syntax errors. • Human judgment was still necessary to meet assignment requirements.	