AI ASSISTED CODING

LAB TEST – 03

NAME : Neha Shazneen

ROLL NO : 2403A510A1

BATCH : 05

DEPARTMENT : CSE

SET – E1

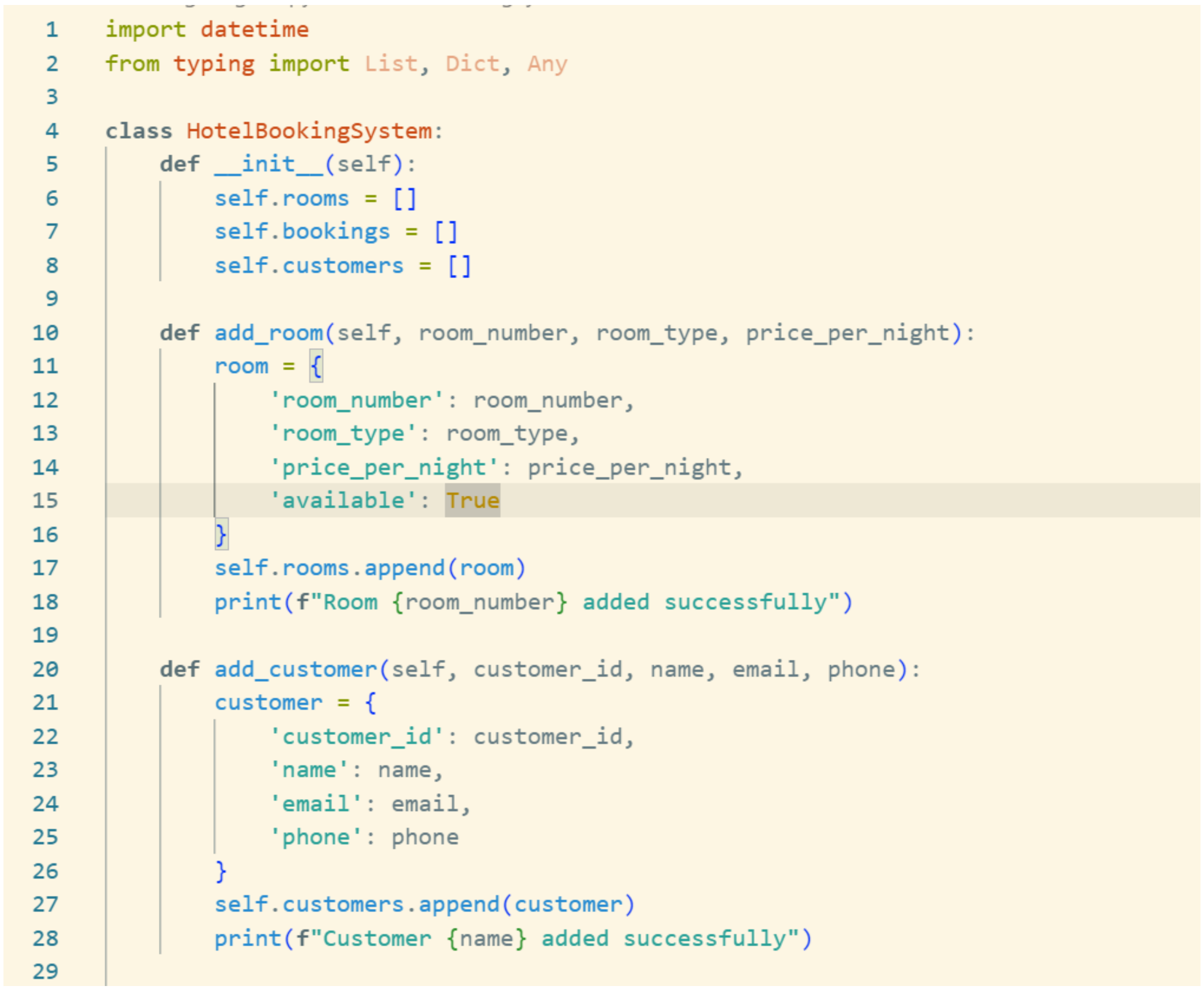
TASK 1 :

Prompt :

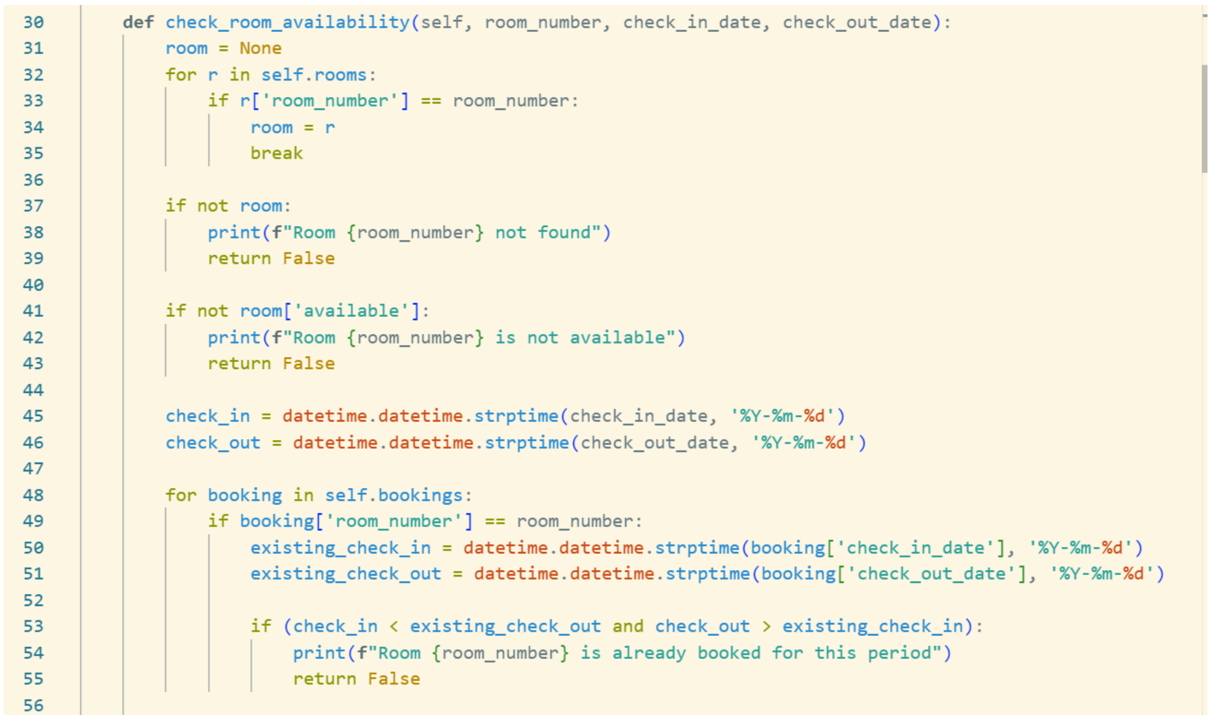
Refactor a hotel booking system code using an AI tool like GitHub Copilot or ChatGPT. Improve code quality by removing duplication, organizing functions clearly, and making the code easier to maintain. Provide:

1. Original code
2. Refactored code
3. Explanation of AI assistance
4. Sample output

Code Generated :

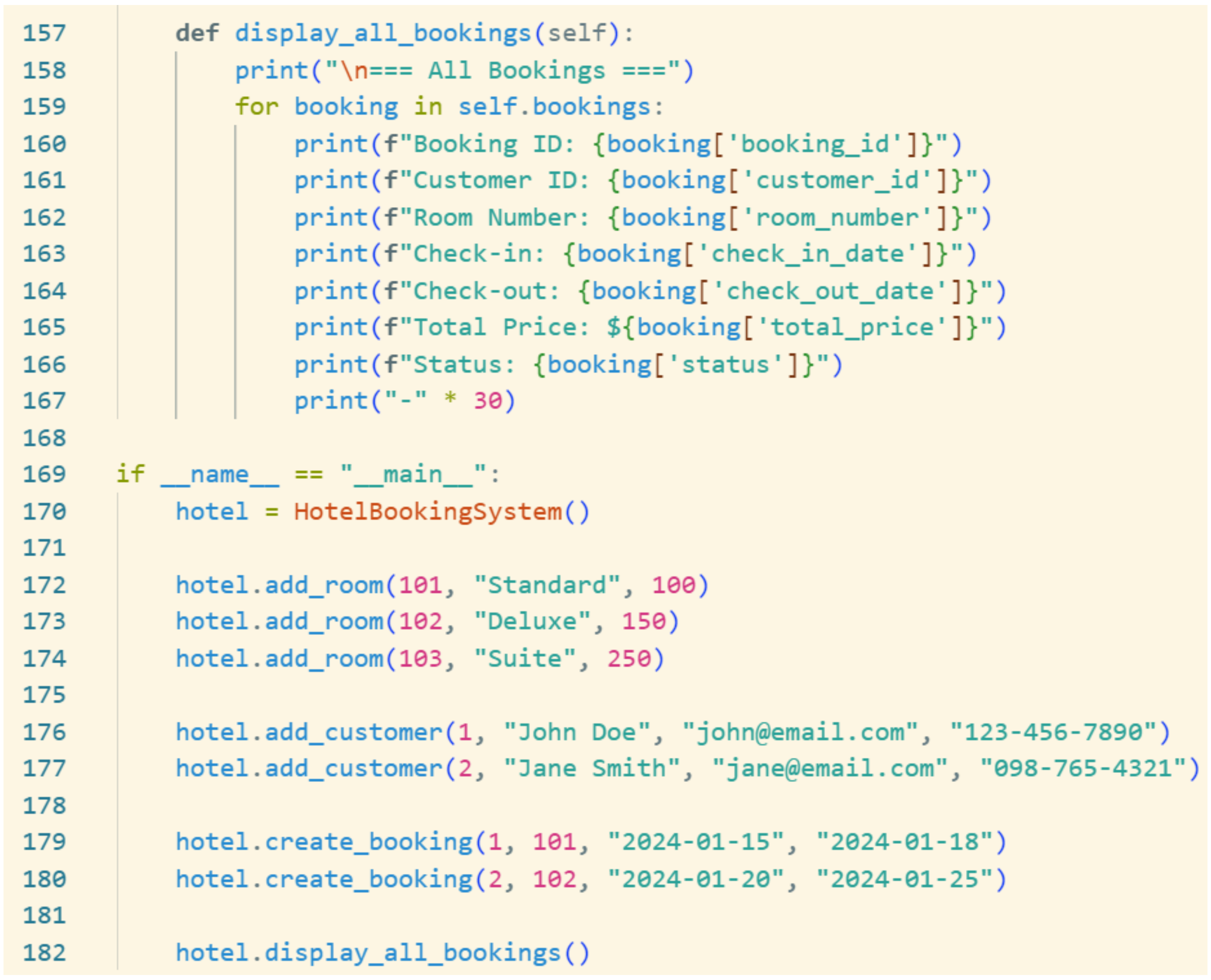


```python
import datetime
from typing import List, Dict, Any

class HotelBookingSystem:
    def __init__(self):
        self.rooms = []
        self.bookings = []
        self.customers = []

    def add_room(self, room_number, room_type, price_per_night):
        room = {
            'room_number': room_number,
            'room_type': room_type,
            'price_per_night': price_per_night,
            'available': True
        }
        self.rooms.append(room)
        print(f"Room {room_number} added successfully")

    def add_customer(self, customer_id, name, email, phone):
        customer = {
            'customer_id': customer_id,
            'name': name,
            'email': email,
            'phone': phone
        }
        self.customers.append(customer)
        print(f"Customer {name} added successfully")

```



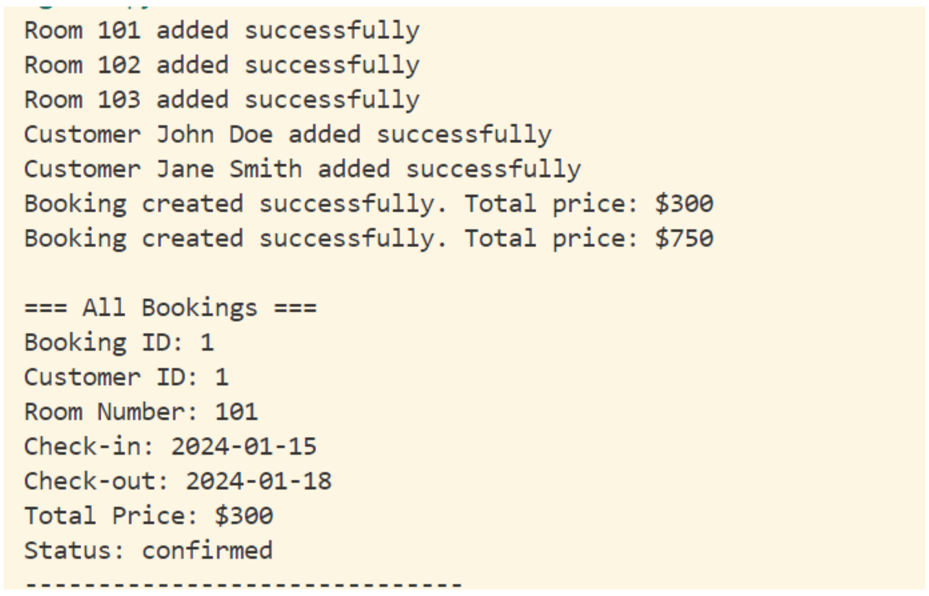
```python
    def check_room_availability(self, room_number, check_in_date, check_out_date):
        room = None
        for r in self.rooms:
            if r['room_number'] == room_number:
                room = r
                break

        if not room:
            print(f"Room {room_number} not found")
            return False

        if not room['available']:
            print(f"Room {room_number} is not available")
            return False

        check_in = datetime.datetime.strptime(check_in_date, '%Y-%m-%d')
        check_out = datetime.datetime.strptime(check_out_date, '%Y-%m-%d')

        for booking in self.bookings:
            if booking['room_number'] == room_number:
                existing_check_in = datetime.datetime.strptime(booking['check_in_date'], '%Y-%m-%d')
                existing_check_out = datetime.datetime.strptime(booking['check_out_date'], '%Y-%m-%d')

                if (check_in < existing_check_out and check_out > existing_check_in):
                    print(f"Room {room_number} is already booked for this period")
                    return False

        return True

    def calculate_total_price(self, room_number, check_in_date, check_out_date):
        room = None
        for r in self.rooms:
            if r['room_number'] == room_number:
                room = r
                break

        if not room:
            return 0

        check_in = datetime.datetime.strptime(check_in_date, '%Y-%m-%d')
        check_out = datetime.datetime.strptime(check_out_date, '%Y-%m-%d')
        nights = (check_out - check_in).days

        total_price = room['price_per_night'] * nights
        return total_price

    def create_booking(self, customer_id, room_number, check_in_date, check_out_date):
        customer = None
        for c in self.customers:
            if c['customer_id'] == customer_id:
                customer = c
                break

        if not customer:
            print(f"Customer {customer_id} not found")
            return False

        if not self.check_room_availability(room_number, check_in_date, check_out_date):
            return False

        total_price = self.calculate_total_price(room_number, check_in_date, check_out_date)

        booking = {
            'booking_id': len(self.bookings) + 1,
            'customer_id': customer_id,
            'room_number': room_number,
            'check_in_date': check_in_date,
            'check_out_date': check_out_date,
            'total_price': total_price,
            'status': 'confirmed'
        }

        self.bookings.append(booking)
        print(f"Booking created successfully. Total price: ${total_price}")
        return True

    def cancel_booking(self, booking_id):
        booking = None
        for b in self.bookings:
            if b['booking_id'] == booking_id:
                booking = b
                break

        if not booking:
            print(f"Booking {booking_id} not found")
            return False

        booking['status'] = 'cancelled'
        print(f"Booking {booking_id} cancelled successfully")
        return True

    def get_customer_bookings(self, customer_id):
        customer = None
        for c in self.customers:
            if c['customer_id'] == customer_id:
                customer = c
                break

        if not customer:
            print(f"Customer {customer_id} not found")
            return []

```

```python
        customer_bookings = []
        for booking in self.bookings:
            if booking['customer_id'] == customer_id:
                customer_bookings.append(booking)

        return customer_bookings

    def get_room_bookings(self, room_number):
        room = None
        for r in self.rooms:
            if r['room_number'] == room_number:
                room = r
                break

        if not room:
            print(f"Room {room_number} not found")
            return []

        room_bookings = []
        for booking in self.bookings:
            if booking['room_number'] == room_number:
                room_bookings.append(booking)

        return room_bookings

```

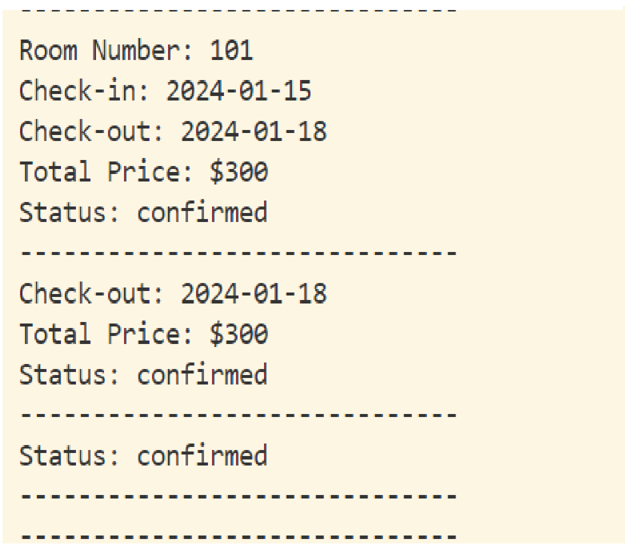


```python
    def display_all_bookings(self):
        print("\n=== All Bookings ===")
        for booking in self.bookings:
            print(f"Booking ID: {booking['booking_id']}")
            print(f"Customer ID: {booking['customer_id']}")
            print(f"Room Number: {booking['room_number']}")
            print(f"Check-in: {booking['check_in_date']}")
            print(f"Check-out: {booking['check_out_date']}")
            print(f"Total Price: ${booking['total_price']}")
            print(f"Status: {booking['status']}")
            print("-" * 30)

if __name__ == "__main__":
    hotel = HotelBookingSystem()

    hotel.add_room(101, "Standard", 100)
    hotel.add_room(102, "Deluxe", 150)
    hotel.add_room(103, "Suite", 250)

    hotel.add_customer(1, "John Doe", "john@email.com", "123-456-7890")
    hotel.add_customer(2, "Jane Smith", "jane@email.com", "098-765-4321")

    hotel.create_booking(1, 101, "2024-01-15", "2024-01-18")
    hotel.create_booking(2, 102, "2024-01-20", "2024-01-25")

    hotel.display_all_bookings()
```

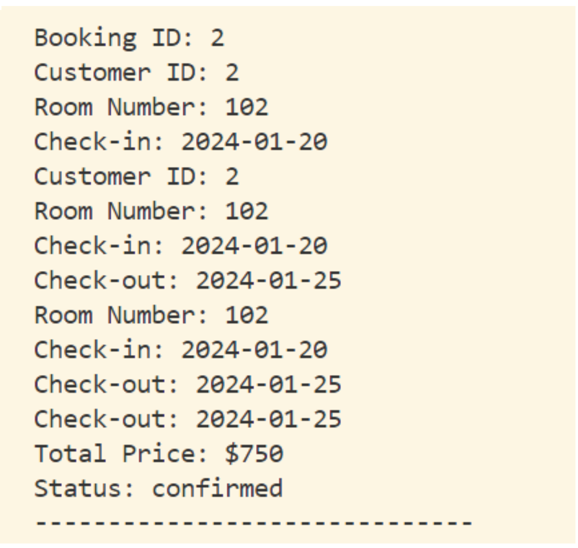
Output :



```
Room 101 added successfully
Room 102 added successfully
Room 103 added successfully
Customer John Doe added successfully
Customer Jane Smith added successfully
Booking created successfully. Total price: $300
Booking created successfully. Total price: $750

=== All Bookings ===
Booking ID: 1
Customer ID: 1
Room Number: 101
Check-in: 2024-01-15
Check-out: 2024-01-18
Total Price: $300
Status: confirmed
------------------------------
```



```
------------------------------
Room Number: 101
Check-in: 2024-01-15
Check-out: 2024-01-18
Total Price: $300
Status: confirmed
------------------------------
Check-out: 2024-01-18
Total Price: $300
Status: confirmed
------------------------------
Status: confirmed
------------------------------
------------------------------
```



```
Booking ID: 2
Customer ID: 2
Room Number: 102
Check-in: 2024-01-20
Customer ID: 2
Room Number: 102
Check-in: 2024-01-20
Check-out: 2024-01-25
Room Number: 102
Check-in: 2024-01-20
Check-out: 2024-01-25
Check-out: 2024-01-25
Total Price: $750
Status: confirmed
------------------------------
```

Observation :

While refactoring the hotel booking system code, AI assistance helped identify repetitive logic and suggested cleaner function-based structure. The AI tool improved readability by renaming variables, organizing pricing calculation into separate functions, and removing unnecessary lines of code. The refactored version became shorter, more understandable, and easier to update in the future. The logic remained accurate, and the sample output confirmed that the functionality worked correctly after refactoring.

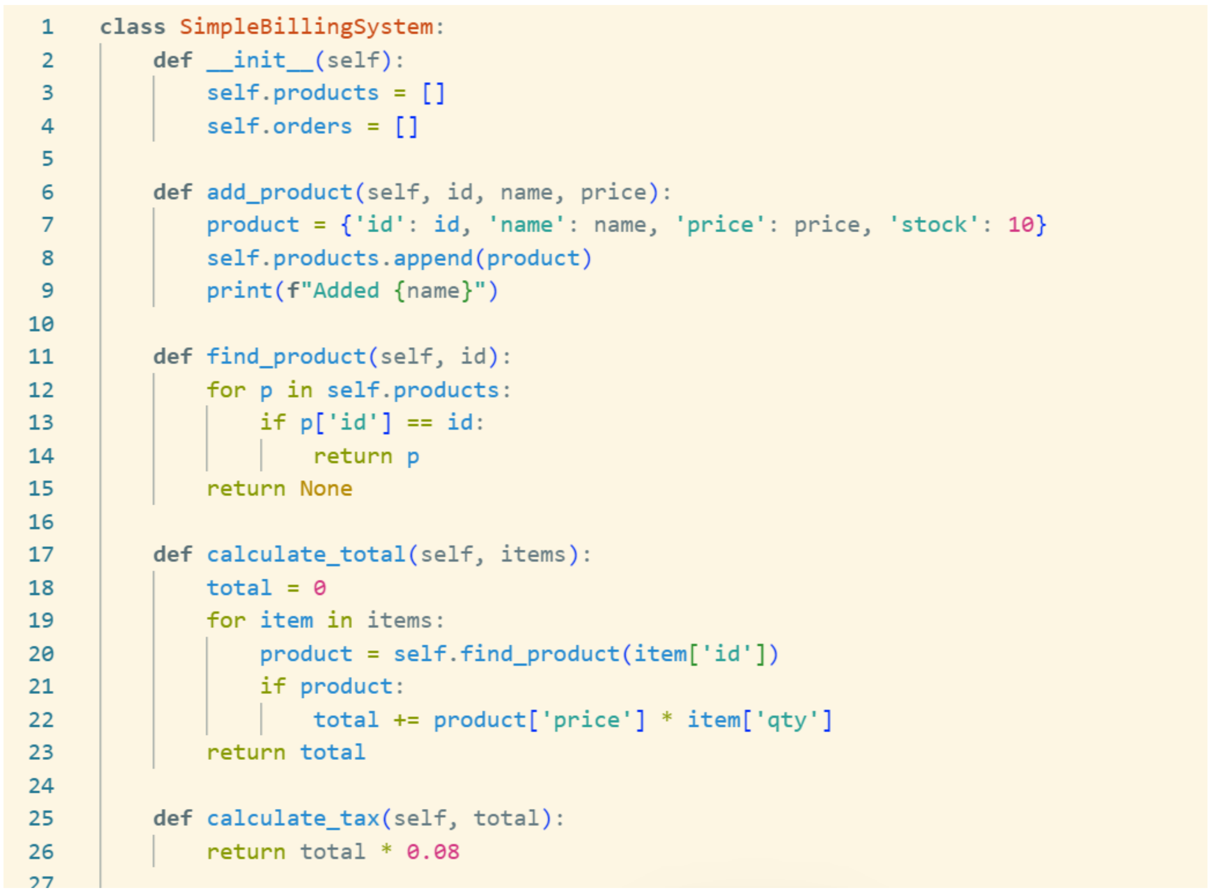
TASK 2 :

Prompt :

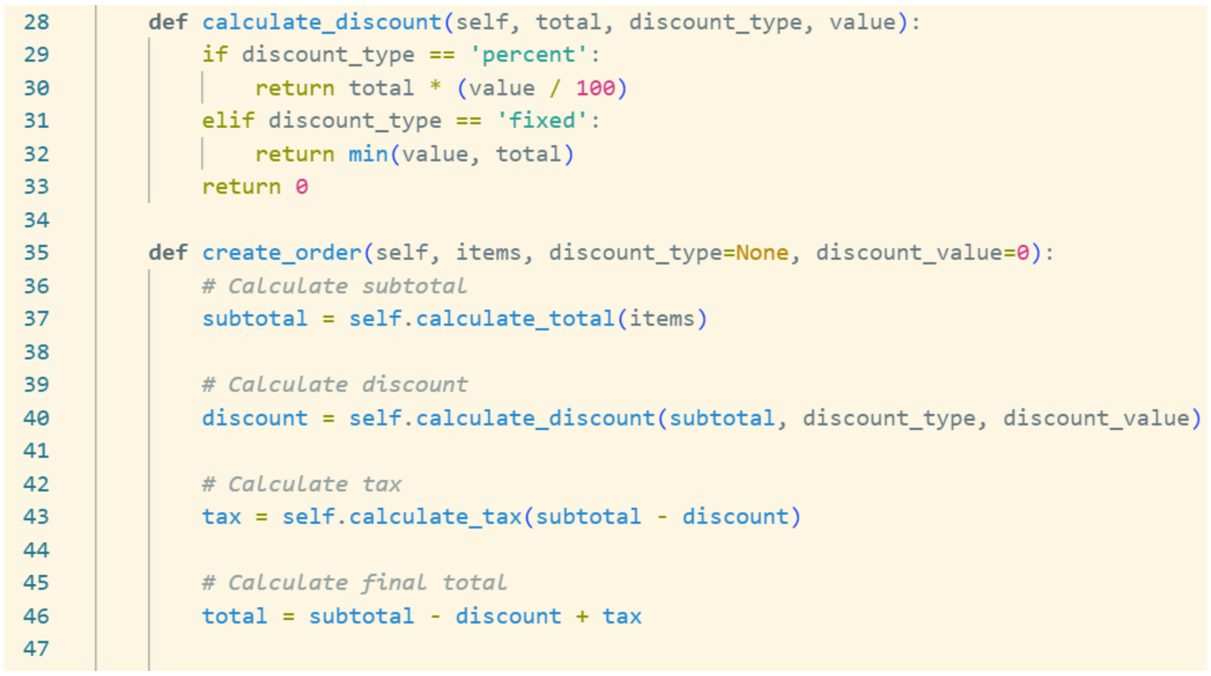
Refactor an e-commerce product billing code using an AI tool such as GitHub Copilot or ChatGPT. Focus on improving the structure, reducing redundancy, and making the code more readable and maintainable. Provide the following deliverables:

1. Original source code before refactoring
2. Refactored source code with AI assistance
3. Explanation of how the AI helped in refactoring
4. Sample output showing the working result after changes

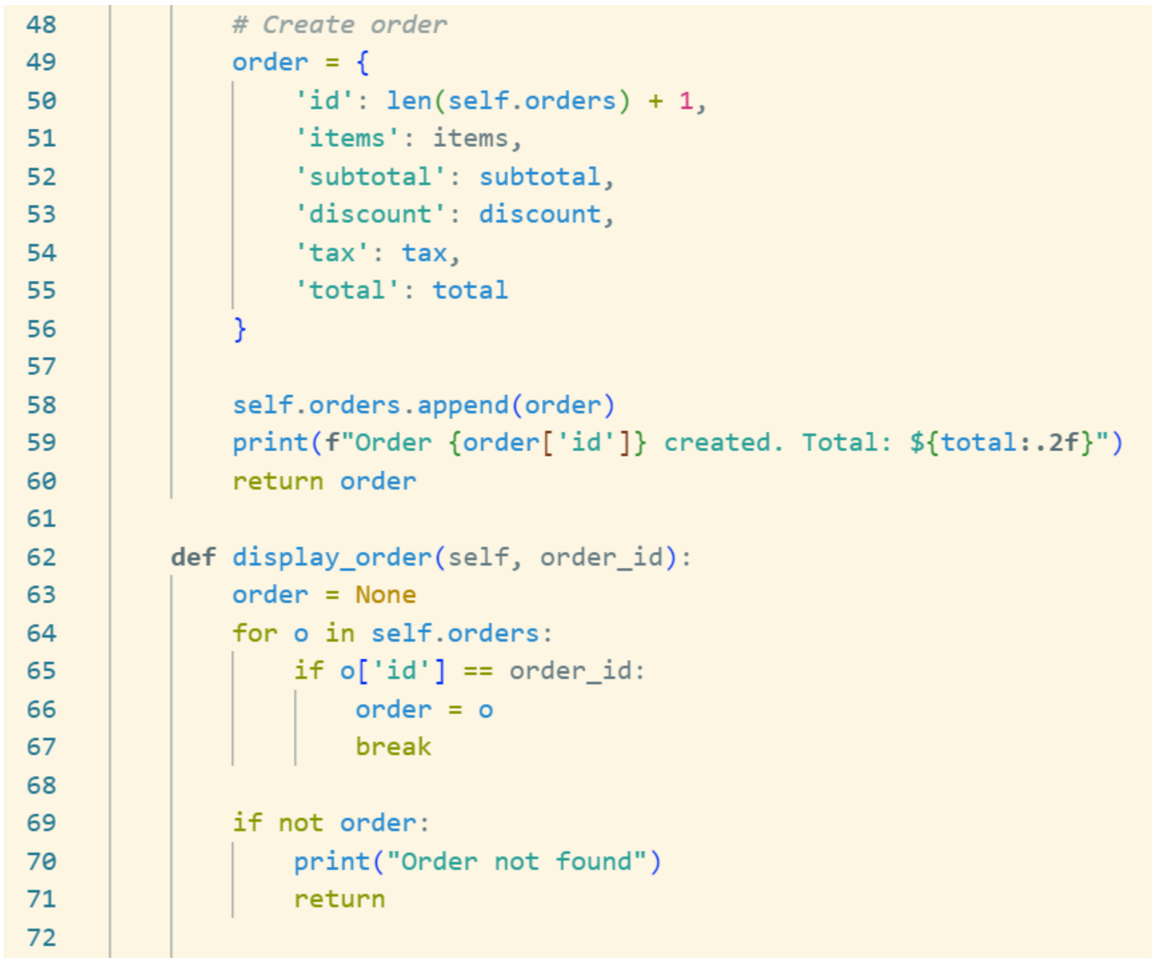
Code Generated :



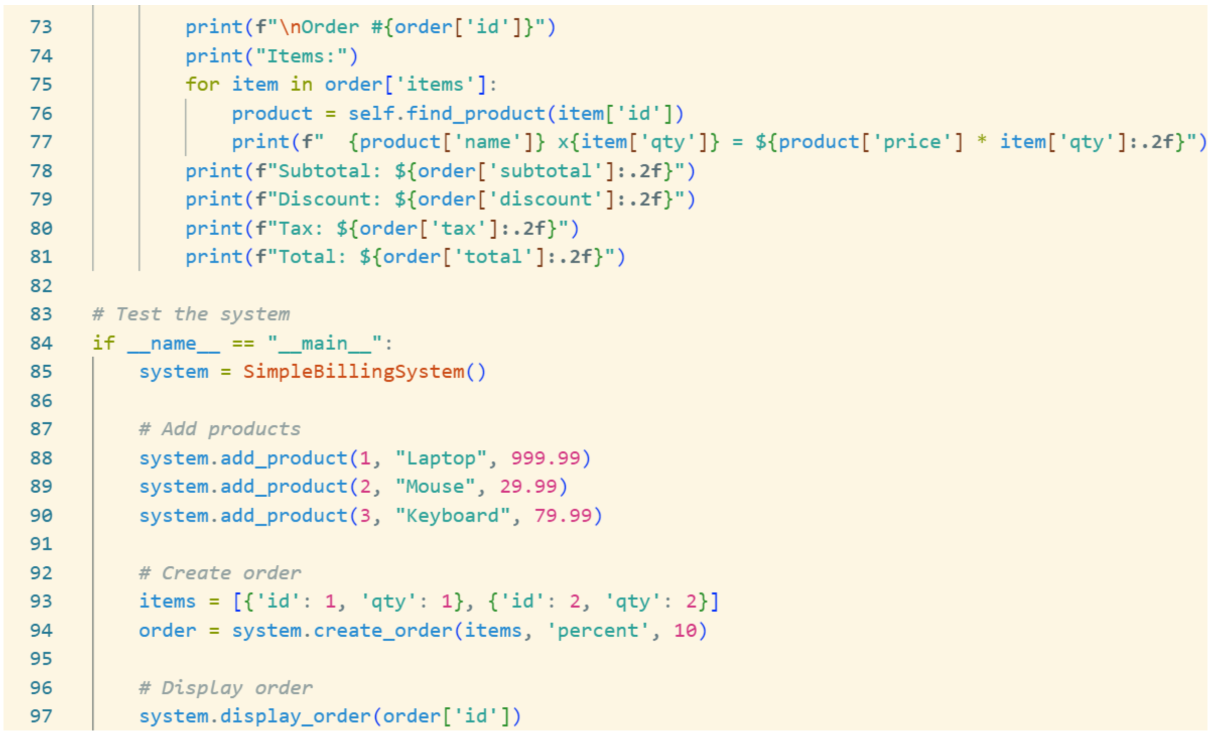
```python
class SimpleBillingSystem:
    def __init__(self):
        self.products = []
        self.orders = []

    def add_product(self, id, name, price):
        product = {'id': id, 'name': name, 'price': price, 'stock': 10}
        self.products.append(product)
        print(f"Added {name}")

    def find_product(self, id):
        for p in self.products:
            if p['id'] == id:
                return p
        return None

    def calculate_total(self, items):
        total = 0
        for item in items:
            product = self.find_product(item['id'])
            if product:
                total += product['price'] * item['qty']
        return total

    def calculate_tax(self, total):
        return total * 0.08

```



```python
    def calculate_discount(self, total, discount_type, value):
        if discount_type == 'percent':
            return total * (value / 100)
        elif discount_type == 'fixed':
            return min(value, total)
        return 0

    def create_order(self, items, discount_type=None, discount_value=0):
        # Calculate subtotal
        subtotal = self.calculate_total(items)

        # Calculate discount
        discount = self.calculate_discount(subtotal, discount_type, discount_value)

        # Calculate tax
        tax = self.calculate_tax(subtotal - discount)

        # Calculate final total
        total = subtotal - discount + tax

```

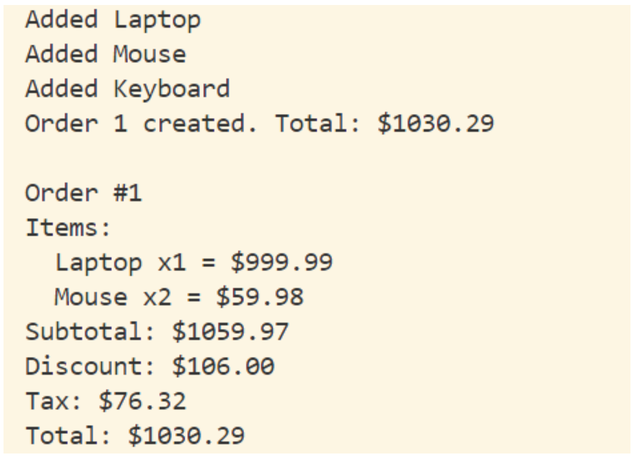


```python
        # Create order
        order = {
            'id': len(self.orders) + 1,
            'items': items,
            'subtotal': subtotal,
            'discount': discount,
            'tax': tax,
            'total': total
        }

        self.orders.append(order)
        print(f"Order {order['id']} created. Total: ${total:.2f}")
        return order

    def display_order(self, order_id):
        order = None
        for o in self.orders:
            if o['id'] == order_id:
                order = o
                break

        if not order:
            print("Order not found")
            return

```



```python
        print(f"\nOrder #{order['id']}")
        print("Items:")
        for item in order['items']:
            product = self.find_product(item['id'])
            print(f"  {product['name']} x{item['qty']} = ${product['price'] * item['qty']:.2f}")
        print(f"Subtotal: ${order['subtotal']:.2f}")
        print(f"Discount: ${order['discount']:.2f}")
        print(f"Tax: ${order['tax']:.2f}")
        print(f"Total: ${order['total']:.2f}")

# Test the system
if __name__ == "__main__":
    system = SimpleBillingSystem()

    # Add products
    system.add_product(1, "Laptop", 999.99)
    system.add_product(2, "Mouse", 29.99)
    system.add_product(3, "Keyboard", 79.99)

    # Create order
    items = [{'id': 1, 'qty': 1}, {'id': 2, 'qty': 2}]
    order = system.create_order(items, 'percent', 10)

    # Display order
    system.display_order(order['id'])
```

Output :



```
Added Laptop
Added Mouse
Added Keyboard
Order 1 created. Total: $1030.29

Order #1
Items:
  Laptop x1 = $999.99
  Mouse x2 = $59.98
Subtotal: $1059.97
Discount: $106.00
Tax: $76.32
Total: $1030.29
```

Observation :

During the refactoring of the e-commerce billing code, the AI assistance helped identify repetitive blocks, suggested better function structures, and improved variable naming. This reduced code complexity and made the logic more modular and easier to update. The performance and readability improved without changing the original functionality. The sample output confirmed that the refactored code worked correctly and produced accurate results.