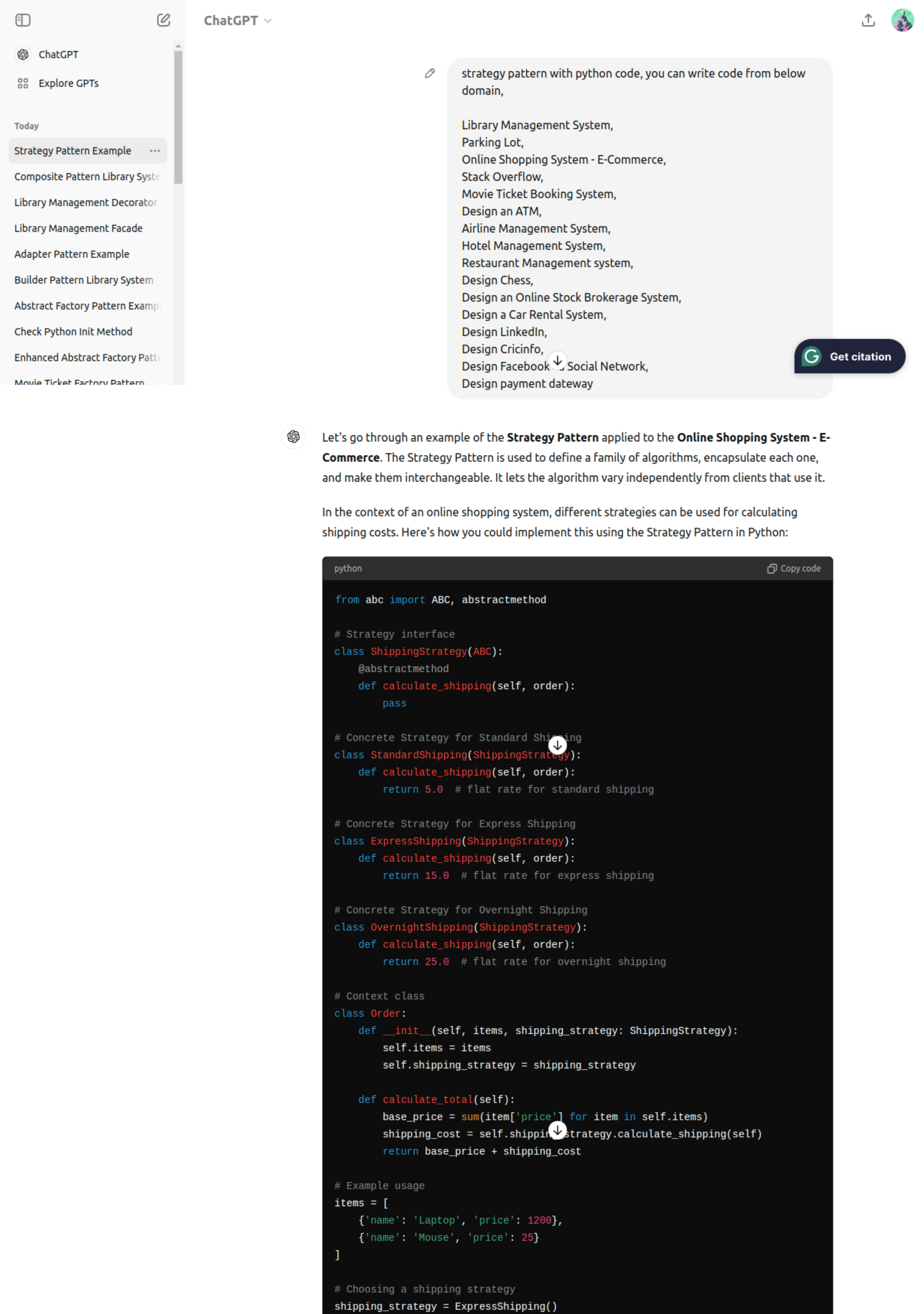
ChatGPT
ChatGPT
Explore GPTs
Today
Strategy Pattern Example
Composite Pattern Library Syste
Library Management Decorator
Library Management Facade
Adapter Pattern Example
Builder Pattern Library System
Abstract Factory Pattern Exampl
Check Python Init Method
Enhanced Abstract Factory Patt
Movie Ticket Factory Pattern
Get citation

strategy pattern with python code, you can write code from below domain,

Library Management System,
Parking Lot,
Online Shopping System - E-Commerce,
Stack Overflow,
Movie Ticket Booking System,
Design an ATM,
Airline Management System,
Hotel Management System,
Restaurant Management system,
Design Chess,
Design an Online Stock Brokerage System,
Design a Car Rental System,
Design LinkedIn,
Design Cricinfo,
Design Facebook - Social Network,
Design payment dateway

Let's go through an example of the **Strategy Pattern** applied to the **Online Shopping System - E-Commerce**. The Strategy Pattern is used to define a family of algorithms, encapsulate each one, and make them interchangeable. It lets the algorithm vary independently from clients that use it.

In the context of an online shopping system, different strategies can be used for calculating shipping costs. Here's how you could implement this using the Strategy Pattern in Python:

```python
from abc import ABC, abstractmethod

# Strategy interface
class ShippingStrategy(ABC):
    @abstractmethod
    def calculate_shipping(self, order):
        pass

# Concrete Strategy for Standard Shipping
class StandardShipping(ShippingStrategy):
    def calculate_shipping(self, order):
        return 5.0  # flat rate for standard shipping

# Concrete Strategy for Express Shipping
class ExpressShipping(ShippingStrategy):
    def calculate_shipping(self, order):
        return 15.0  # flat rate for express shipping

# Concrete Strategy for Overnight Shipping
class OvernightShipping(ShippingStrategy):
    def calculate_shipping(self, order):
        return 25.0  # flat rate for overnight shipping

# Context class
class Order:
    def __init__(self, items, shipping_strategy: ShippingStrategy):
        self.items = items
        self.shipping_strategy = shipping_strategy

    def calculate_total(self):
        base_price = sum(item['price'] for item in self.items)
        shipping_cost = self.shipping_strategy.calculate_shipping(self)
        return base_price + shipping_cost

# Example usage
items = [
    {'name': 'Laptop', 'price': 1200},
    {'name': 'Mouse', 'price': 25}
]

# Choosing a shipping strategy
shipping_strategy = ExpressShipping()
```

```
order = Order(items, shipping_strategy)

print(f"Total order cost with express shipping: ${order.calculate_total()}")

# Switching strategy to Overnight Shipping
order.shipping_strategy = OvernightShipping()
print(f"Total order cost with overnight shipping: ${order.calculate_total()}")
```

Explanation:

1. **ShippingStrategy (Strategy Interface)**: An abstract class with a method `calculate_shipping` that needs to be implemented by concrete strategies.
2. **Concrete Strategies**: `StandardShipping`, `ExpressShipping`, and `OvernightShipping` each implement the `calculate_shipping` method differently.
3. **Order (Context Class)**: Uses a `ShippingStrategy` object to calculate the total cost of the order. The strategy can be changed dynamically, demonstrating the flexibility provided by the Strategy Pattern.

In this example, the shipping cost calculation strategy can be changed easily by replacing the `ShippingStrategy` implementation, without altering the `Order` class's code. This allows for greater flexibility and easier maintenance of the shipping cost logic.

Upgrade plan
Get GPT-4, DALL·E, and more

Message ChatGPT

ChatGPT can make mistakes. Check important info.

?