

Task #1

Prompt:

start a python class called Student with attributes : name, Roll_no and marks.and also add methods like displaying details and checking if the marks are above average

ai.py +14 -0

The `Student` class has been created in `ai.py` with attributes `name`, `roll_no`, and `marks`. It includes methods to display details and check if the marks are above average. Let me know if you want to add more features or see usage examples.




make sure to check the code again and include the use of method of displaying details and checking if the marks are above average with the use of if else condition

ai.py +9 -0

The code now includes an example that creates a `Student` object, displays its

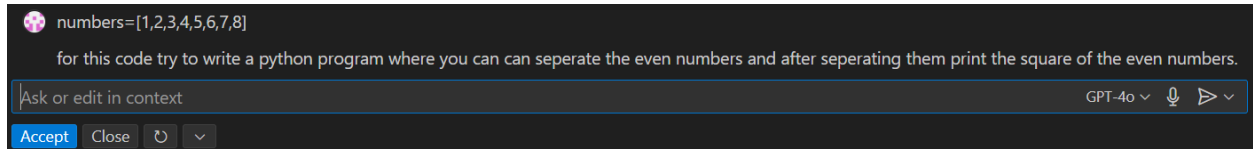
Code:

 ai.py > ...

```
1  # Student class definition
2  class Student:
3      def __init__(self, name, roll_no, marks):
4          self.name = name
5          self.roll_no = roll_no
6          self.marks = marks
7
8      def display_details(self):
9          print(f"Name: {self.name}")
10         print(f"Roll No: {self.roll_no}")
11         print(f"Marks: {self.marks}")
12
13         def is_above_average(self, average=50):
14             return self.marks > average
15
16     # Example usage
17     if __name__ == "__main__":
18         student1 = Student("Alice", 101, 75)
19         student1.display_details()
20         if student1.is_above_average():
21             print("Marks are above average.")
22         else:
23             print("Marks are not above average.")
24
```

Task #2

Prompt:



Code:

```
# List of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8]

# Separate even numbers and calculate their squares
even_numbers = [num for num in numbers if num % 2 == 0]
squared_evens = [num ** 2 for num in even_numbers]

# Print the squares of even numbers
print("Squares of even numbers:", squared_evens)
```

Task #3

Prompt:

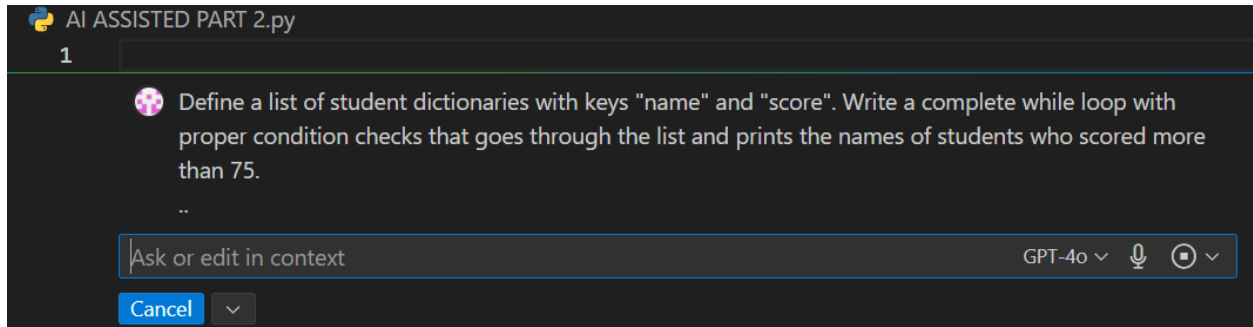
🔗 create a class called bank account with attributes such as account_holder and balance . also include methods like deposit(), withdrawal, and checking balance make sure that the user doesnt overdraw from it and the functional class should complete methods using if conditions and self attributes.

Code:

```
AI ASSISTED PART 2.py > ...
1  class BankAccount:
2      def __init__(self, account_holder, initial_balance=0):
3          self.account_holder = account_holder
4          self.balance = initial_balance
5
6      def deposit(self, amount):
7          if amount > 0:
8              self.balance += amount
9              print(f"Deposited {amount}. New balance: {self.balance}")
10         else:
11             print("Deposit amount must be positive.")
12
13     def withdrawal(self, amount):
14         if amount > 0:
15             if self.balance >= amount:
16                 self.balance -= amount
17                 print(f"Withdrew {amount}. New balance: {self.balance}")
18             else:
19                 print("Insufficient funds. Withdrawal denied.")
20         else:
21             print("Withdrawal amount must be positive.")
22
23     def check_balance(self):
24         print(f"Account holder: {self.account_holder}, Balance: {self.balance}")
25
26 # Example usage:
27 # account = BankAccount("John Doe", 100)
28 # account.deposit(50)
29 # account.withdrawal(30)
30 # account.check_balance()
```

Task #4

Prompt:



code

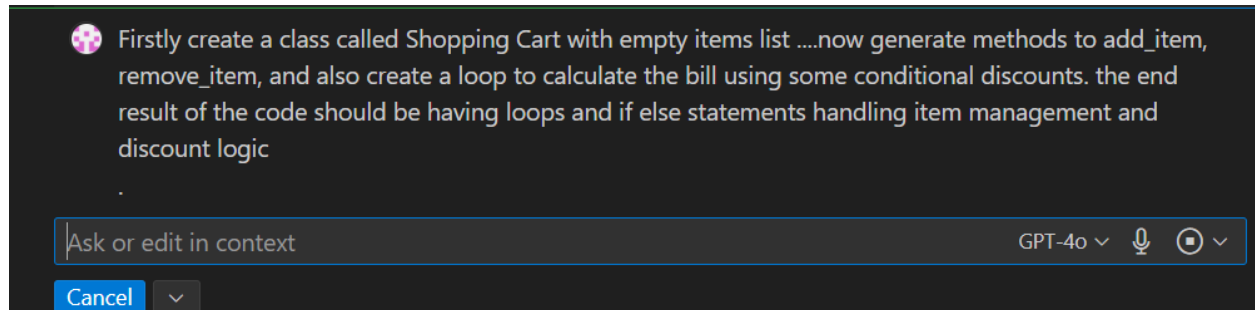
```
# Define a list of student dictionaries
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 72},
    {"name": "Charlie", "score": 90},
    {"name": "Diana", "score": 65},
    {"name": "Eve", "score": 78}
]

# Initialize an index for the while loop
index = 0

# Loop through the list of students
while index < len(students):
    # Check if the student's score is greater than 75
    if students[index]["score"] > 75:
        # Print the student's name
        print(students[index]["name"])
    # Increment the index
    index += 1
```

task #5

Prompt:



Code:

AI ASSISTED PART 2.py > ...

```
1 class ShoppingCart:
2     def __init__(self):
3         # Initialize an empty list to store items
4         self.items = []
5
6     def add_item(self, item_name, price, quantity=1):
7         # Add an item to the shopping cart
8         self.items.append({
9             'name': item_name,
10            'price': price,
11            'quantity': quantity
12        })
13        print(f"Added {quantity} x {item_name} to the cart.")
14
15    def remove_item(self, item_name):
16        # Remove an item from the shopping cart
17        for item in self.items:
18            if item['name'] == item_name:
19                self.items.remove(item)
20                print(f"Removed {item_name} from the cart.")
21                return
22        print(f"Item {item_name} not found in the cart.")
23
24    def calculate_bill(self):
25        # Calculate the total bill with conditional discounts
26        total = 0
27        for item in self.items:
28            item_total = item['price'] * item['quantity']
29            print(f"{item['quantity']} x {item['name']} @ {item['price']} each = {item_total}")
30            total += item_total
31
32        # Apply discounts based on conditions
33        if total > 100:
34            discount = total * 0.1 # 10% discount for bills over $100
35            print(f"Discount applied: -{discount}")
36            total -= discount
37        elif len(self.items) > 5:
38            discount = 5 # Flat $5 discount for more than 5 items
39            print(f"Discount applied: -{discount}")
40            total -= discount
41
42        print(f"Total bill: {total}")
43        return total
44
45
46 # Example usage
47 if __name__ == "__main__":
48     cart = ShoppingCart()
49     cart.add_item("Apple", 1.5, 4)
50     cart.add_item("Banana", 0.5, 10)
51     cart.add_item("Milk", 3.0, 2)
52     cart.remove_item("Banana")
53     cart.add_item("Bread", 2.5, 1)
54     cart.calculate_bill()
```