Assignment 6.4

Name:B.Sanjana

H.T.No:2303A52306

Batch:43

Task 1: Student Performance Evaluation System

Scenario

You are building a simple academic management module for a university system where student performance needs to be evaluated automatically.

Task Description

Create the skeleton of a Python class named Student with the attributes:

- name
- roll_number
- marks

Write only the class definition and attribute initialization.

Then, using GitHub Copilot, prompt the tool to complete:

- A method to display student details
- A method that checks whether the student's marks are above the class average and returns an appropriate message

Use comments or partial method names to guide Copilot for code completion.

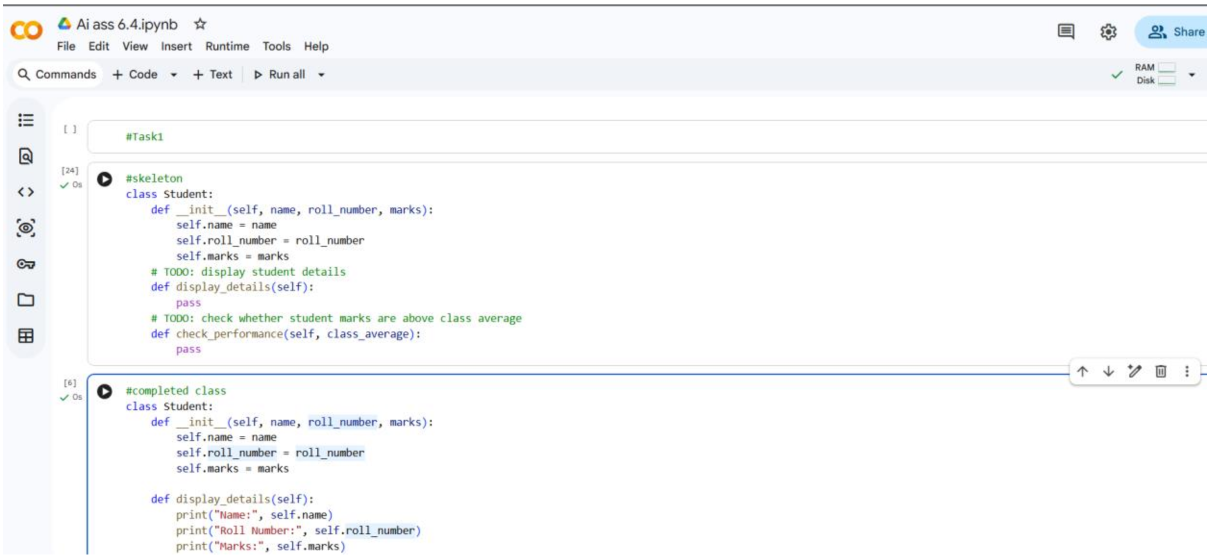
Expected Outcome

- A completed Student class with Copilot-generated methods
- Proper use of:

o self attributes

o Conditional statements (if-else)

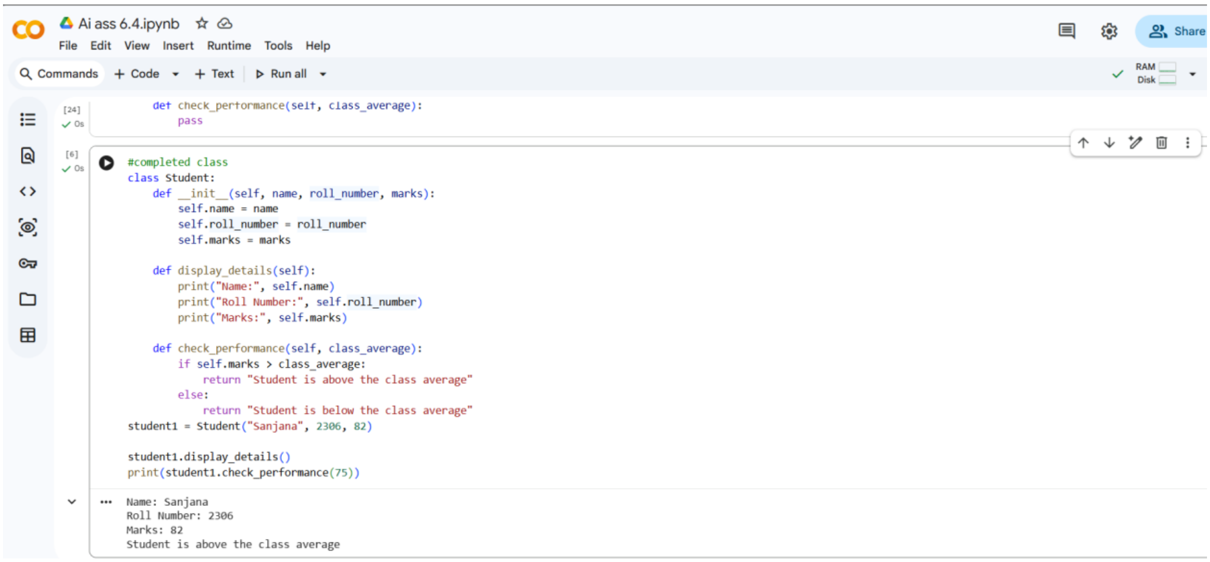
- Sample output showing student details and performance status


Ai ass 6.4.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk

```
[ ]
    #Task1
```

```
[24]
#skeleton
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
    # TODO: display student details
    def display_details(self):
        pass
    # TODO: check whether student marks are above class average
    def check_performance(self, class_average):
        pass
```

```
[6]
#completed class
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)
```


Ai ass 6.4.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk

```
[24]
    def check_performance(self, class_average):
        pass
```

```
[6]
#completed class
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)

    def check_performance(self, class_average):
        if self.marks > class_average:
            return "Student is above the class average"
        else:
            return "Student is below the class average"
student1 = Student("Sanjana", 2306, 82)

student1.display_details()
print(student1.check_performance(75))
```

```
Name: Sanjana
Roll Number: 2306
Marks: 82
Student is above the class average
```

Task 2: Data Processing in a Monitoring System

Scenario

You are working on a basic data monitoring script where sensor readings are collected as numbers. Only even readings need further processing.

Task Description

Write the initial part of a for loop to iterate over a list of integers representing sensor readings.

Add a comment prompt instructing GitHub Copilot to:

- Identify even numbers
- Calculate their square
- Print the result in a readable format

Allow Copilot to complete the remaining loop logic.

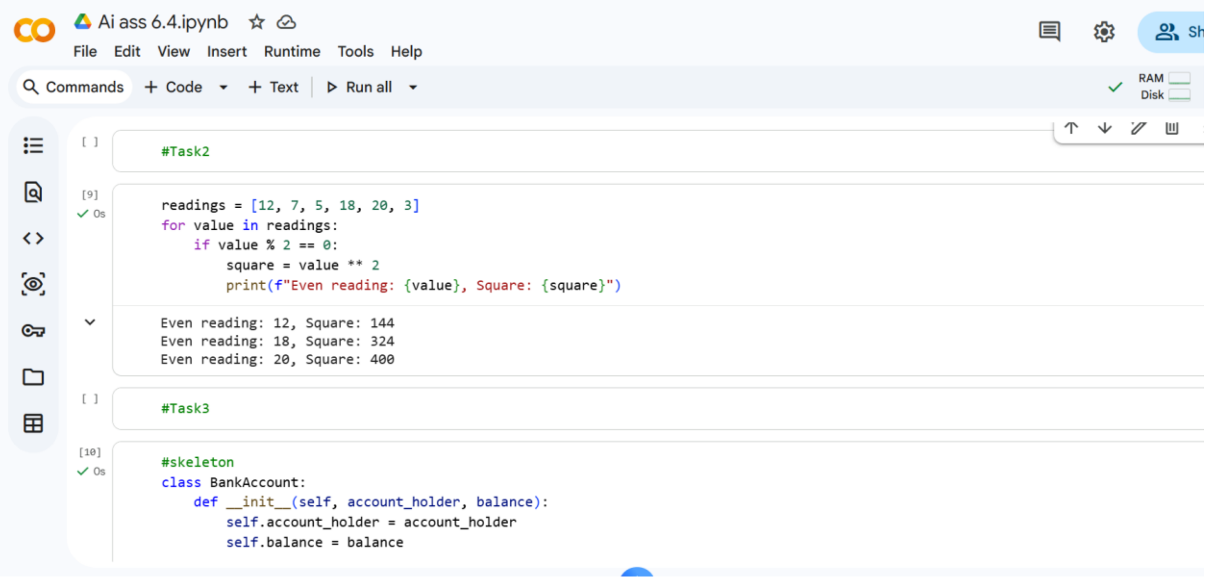
Expected Outcome

- A complete for loop generated by Copilot
- Use of:

o Modulus operator to identify even numbers

o Conditional statements

- Correct and formatted output for valid inputs


Ai ass 6.4.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM Disk

```
#Task2
```

```
readings = [12, 7, 5, 18, 20, 3]
for value in readings:
    if value % 2 == 0:
        square = value ** 2
        print(f"Even reading: {value}, Square: {square}")
```

```
Even reading: 12, Square: 144
Even reading: 18, Square: 324
Even reading: 20, Square: 400
```

```
#Task3
```

```
#skeleton
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
```

Task 3: Banking Transaction Simulation

Scenario

You are developing a basic banking module that handles deposits and withdrawals for customers.

Task Description

Create the structure of a Python class named BankAccount with attributes:

- account_holder
- balance

Use GitHub Copilot to complete methods for:

- Depositing money
- Withdrawing money
- Preventing withdrawals when the balance is insufficient

Guide Copilot using method names and short comments.

Expected Outcome

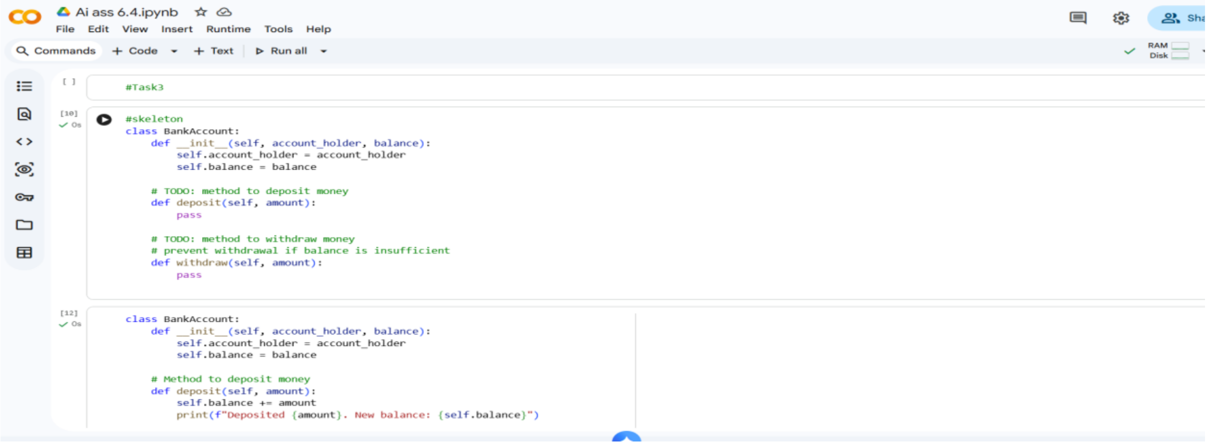
- A fully functional BankAccount class
- Copilot-generated methods using:

o if-else conditions

o Class attributes via self

- Proper handling of invalid withdrawal attempts with user-friendly

Messages


Ai ass 6.4.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM Disk

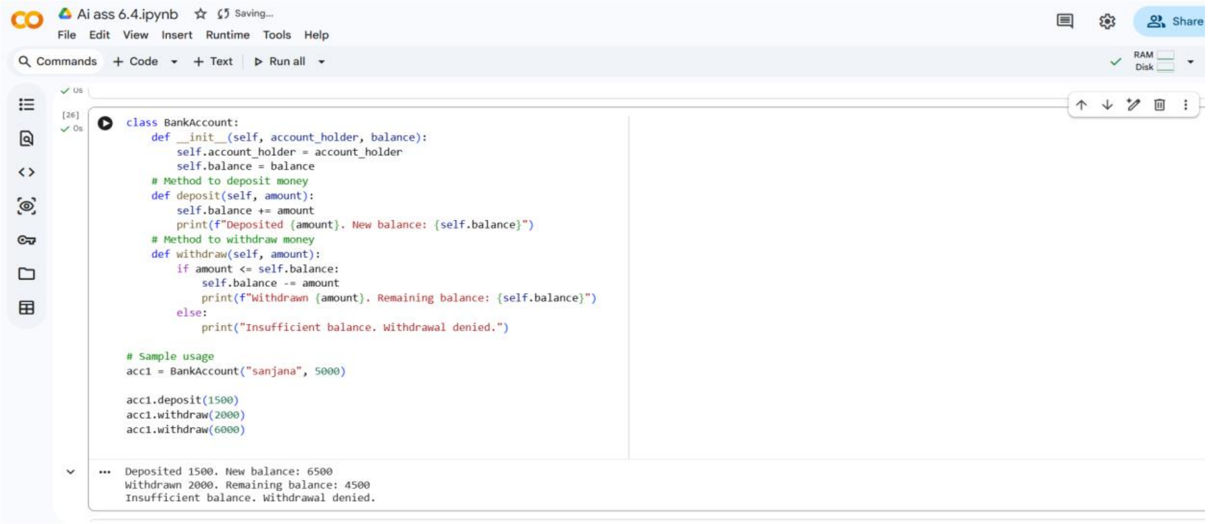
```
#Task3
```

[10]

```
#skeleton
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # TODO: method to deposit money
    def deposit(self, amount):
        pass

    # TODO: method to withdraw money
    # prevent withdrawal if balance is insufficient
    def withdraw(self, amount):
        pass
```

[12]

```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # Method to deposit money
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")
```


Ai ass 6.4.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk

[26]

```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
    # Method to deposit money
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")
    # Method to withdraw money
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn {amount}. Remaining balance: {self.balance}")
        else:
            print("Insufficient balance. Withdrawal denied.")

# Sample usage
acc1 = BankAccount("sanjana", 5000)

acc1.deposit(1500)
acc1.withdraw(2000)
acc1.withdraw(6000)
```

```
Deposited 1500. New balance: 6500
Withdrawn 2000. Remaining balance: 4500
Insufficient balance. Withdrawal denied.
```

Task 4: Student Scholarship Eligibility Check

Scenario

A university wants to identify students eligible for a merit-based

scholarship based on their scores.

Task Description

Define a list of dictionaries where each dictionary represents a student

with:

- name

- score

Write the initialization and list structure yourself.

Then, prompt GitHub Copilot to generate a while loop that:

- Iterates through the list

- Prints the names of students who scored more than 75

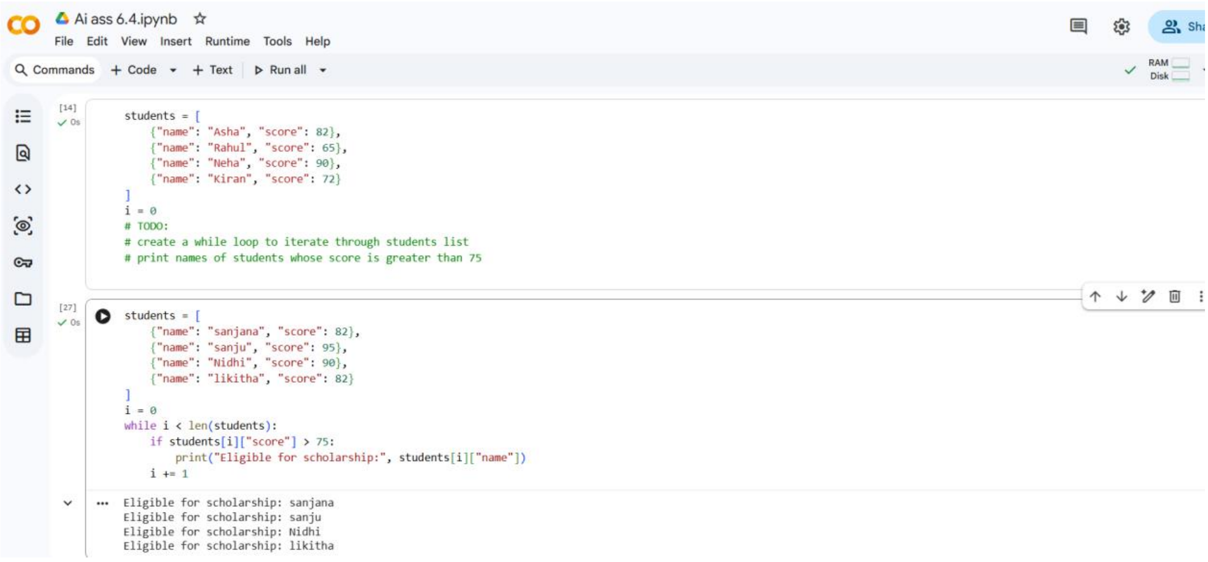
Use comments to guide Copilot's code completion.

Expected Outcome

- A complete while loop generated by Copilot

- Correct index handling and condition checks

- Cleanly formatted output listing eligible students



```
students = [
    {"name": "Asha", "score": 82},
    {"name": "Rahul", "score": 65},
    {"name": "Neha", "score": 90},
    {"name": "Kiran", "score": 72}
]
i = 0
# TODO:
# create a while loop to iterate through students list
# print names of students whose score is greater than 75
```

```
students = [
    {"name": "sanjana", "score": 82},
    {"name": "sanju", "score": 95},
    {"name": "Nidhi", "score": 90},
    {"name": "likitha", "score": 82}
]
i = 0
while i < len(students):
    if students[i]["score"] > 75:
        print("Eligible for scholarship:", students[i]["name"])
    i += 1
```

```
Eligible for scholarship: sanjana
Eligible for scholarship: sanju
Eligible for scholarship: Nidhi
Eligible for scholarship: likitha
```

Task 5: Online Shopping Cart Module

Scenario

You are designing a simplified shopping cart system for an e-commerce website that supports item management and discount calculation.

Task Description

Begin writing a Python class named ShoppingCart with:

- An empty list to store items (each item may include name, price, quantity)

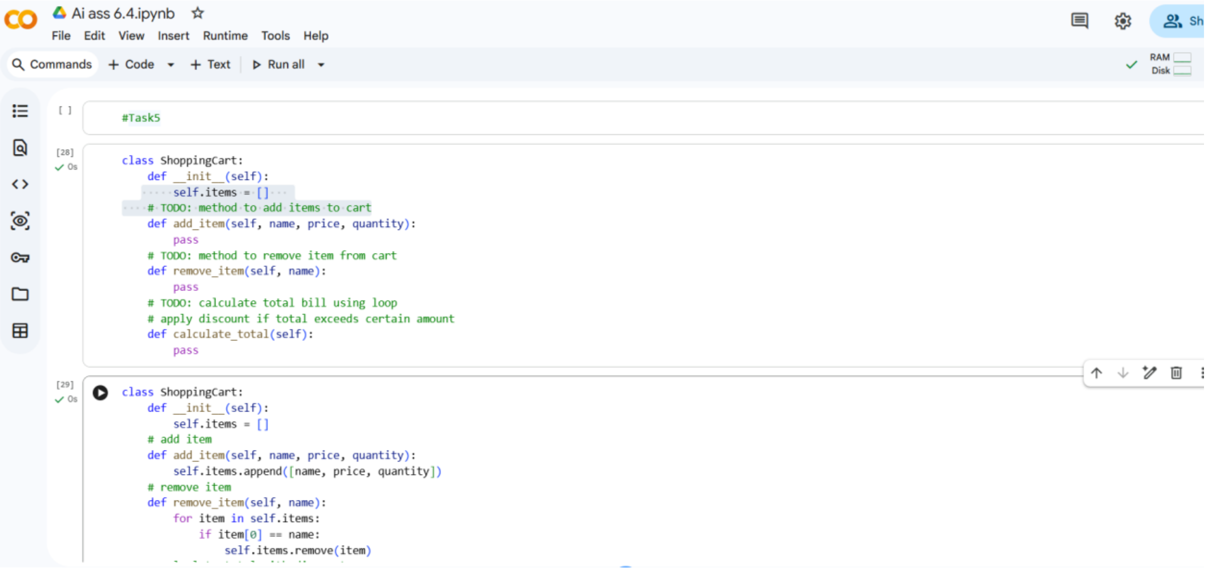
Use GitHub Copilot to generate methods that:

- Add items to the cart
- Remove items from the cart
- Calculate the total bill using a loop
- Apply conditional discounts (e.g., discount if total exceeds a certain amount)

Use meaningful comments and method names to guide Copilot.

Expected Outcome

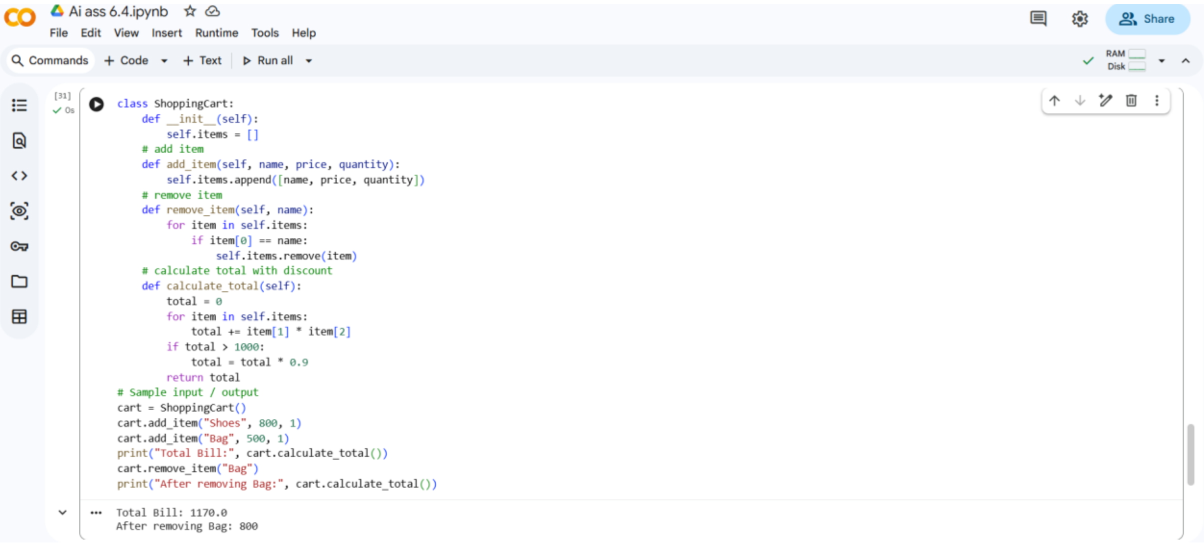
- A fully implemented ShoppingCart class
- Copilot-generated loops and conditional logic
- Correct handling of item addition, removal, and discount calculation
- Sample input/output demonstrating cart functionality



```
#Task5
```

```
class ShoppingCart:
    def __init__(self):
        self.items = []
    # TODO: method to add items to cart
    def add_item(self, name, price, quantity):
        pass
    # TODO: method to remove item from cart
    def remove_item(self, name):
        pass
    # TODO: calculate total bill using loop
    # apply discount if total exceeds certain amount
    def calculate_total(self):
        pass
```

```
class ShoppingCart:
    def __init__(self):
        self.items = []
    # add item
    def add_item(self, name, price, quantity):
        self.items.append([name, price, quantity])
    # remove item
    def remove_item(self, name):
        for item in self.items:
            if item[0] == name:
                self.items.remove(item)
```



```
class ShoppingCart:
    def __init__(self):
        self.items = []
    # add item
    def add_item(self, name, price, quantity):
        self.items.append([name, price, quantity])
    # remove item
    def remove_item(self, name):
        for item in self.items:
            if item[0] == name:
                self.items.remove(item)
    # calculate total with discount
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item[1] * item[2]
        if total > 1000:
            total = total * 0.9
        return total
# Sample input / output
cart = ShoppingCart()
cart.add_item("Shoes", 800, 1)
cart.add_item("Bag", 500, 1)
print("Total Bill:", cart.calculate_total())
cart.remove_item("Bag")
print("After removing Bag:", cart.calculate_total())
```

```
Total Bill: 1170.0
After removing Bag: 800
```