AI ASSISSTANT CODING LAB-6.3

Name:M.Shashidhar

Roll no:2303a52291 Batch :42

Task-1:**Student Performance Evaluation System**

Scenario

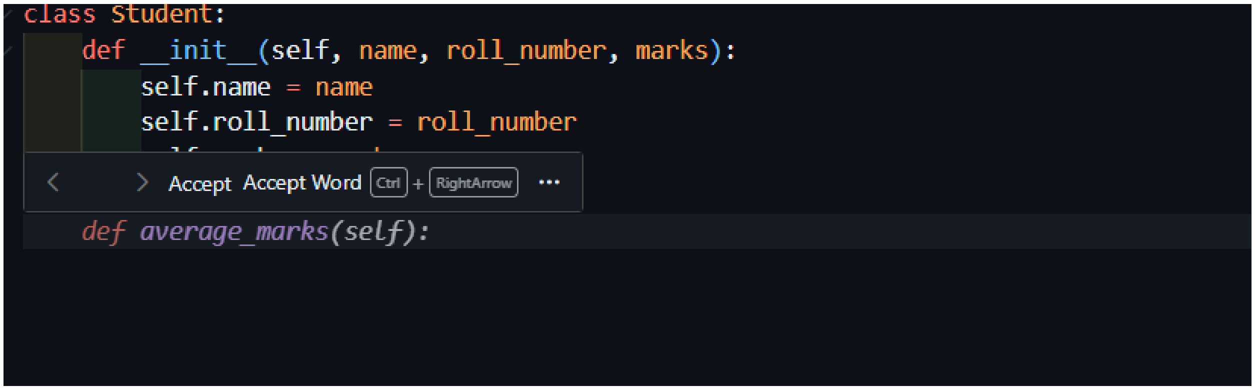
Automate student performance evaluation using a Python class.

prompt:

# display student details

# check whether marks are above average and return message

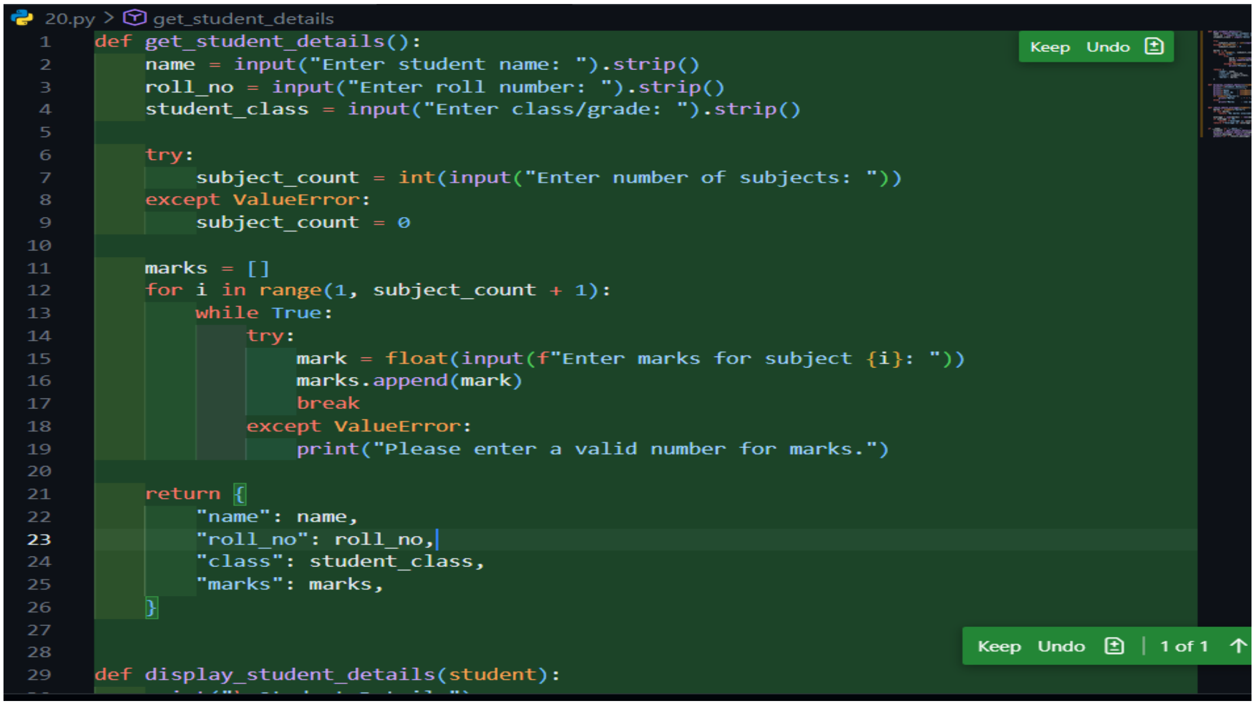
INTIAL SKELETON



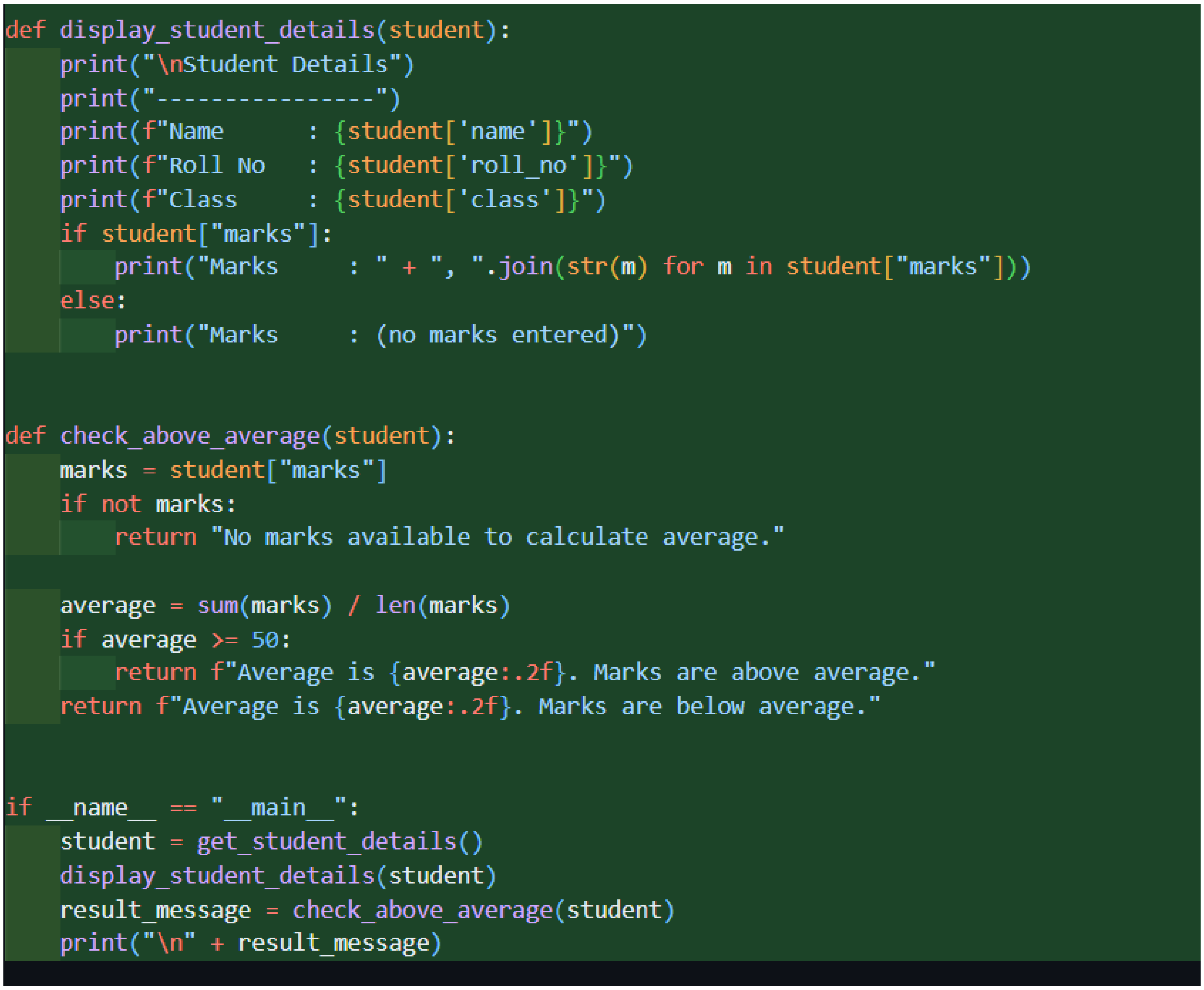
```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
```

```
    def average_marks(self):
```

Generated code

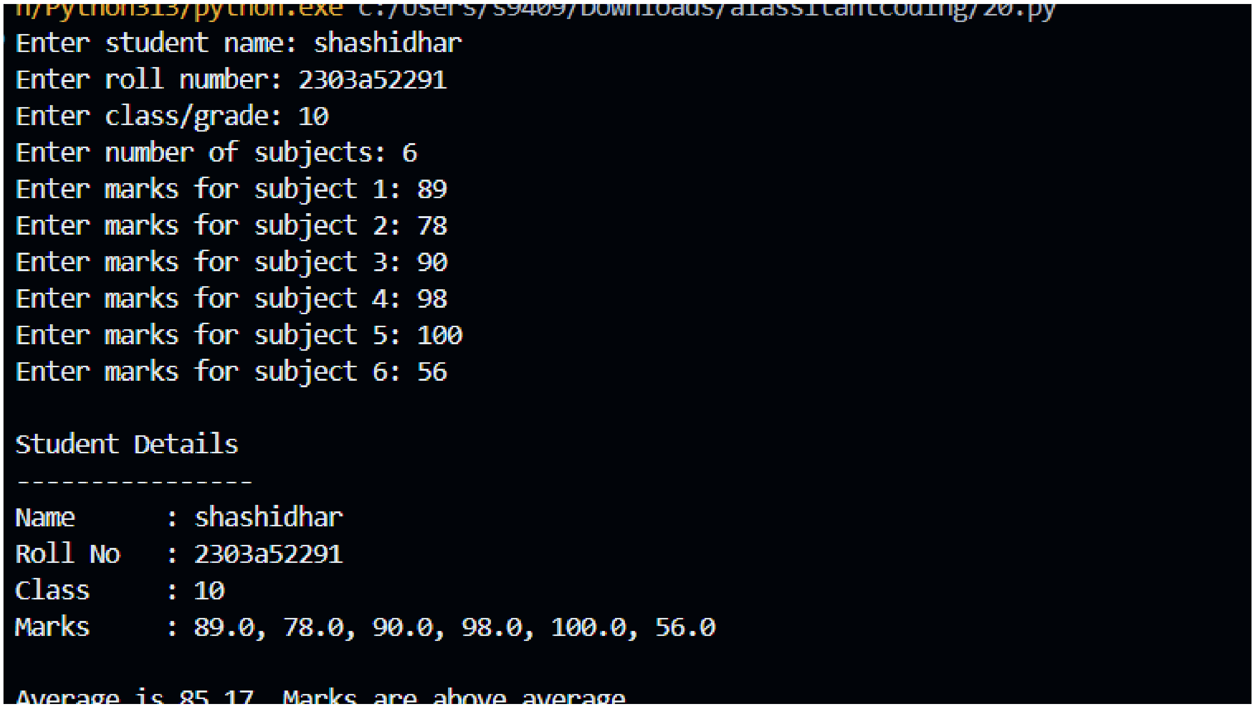


```python
def get_student_details():
    name = input("Enter student name: ").strip()
    roll_no = input("Enter roll number: ").strip()
    student_class = input("Enter class/grade: ").strip()

    try:
        subject_count = int(input("Enter number of subjects: "))
    except ValueError:
        subject_count = 0

    marks = []
    for i in range(1, subject_count + 1):
        while True:
            try:
                mark = float(input(f"Enter marks for subject {i}: "))
                marks.append(mark)
                break
            except ValueError:
                print("Please enter a valid number for marks.")

    return {
        "name": name,
        "roll_no": roll_no,
        "class": student_class,
        "marks": marks,
    }


def display_student_details(student):
```



```python
def display_student_details(student):
    print("\nStudent Details")
    print("----------------")
    print(f"Name      : {student['name']}")
    print(f"Roll No   : {student['roll_no']}")
    print(f"Class     : {student['class']}")
    if student["marks"]:
        print("Marks     : " + ", ".join(str(m) for m in student["marks"]))
    else:
        print("Marks     : (no marks entered)")


def check_above_average(student):
    marks = student["marks"]
    if not marks:
        return "No marks available to calculate average."

    average = sum(marks) / len(marks)
    if average >= 50:
        return f"Average is {average:.2f}. Marks are above average."
    return f"Average is {average:.2f}. Marks are below average."


if __name__ == "__main__":
    student = get_student_details()
    display_student_details(student)
    result_message = check_above_average(student)
    print("\n" + result_message)
```

Output:



```
n/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/20.py
Enter student name: shashidhar
Enter roll number: 2303a52291
Enter class/grade: 10
Enter number of subjects: 6
Enter marks for subject 1: 89
Enter marks for subject 2: 78
Enter marks for subject 3: 90
Enter marks for subject 4: 98
Enter marks for subject 5: 100
Enter marks for subject 6: 56

Student Details
----------------
Name      : shashidhar
Roll No   : 2303a52291
Class     : 10
Marks     : 89.0, 78.0, 90.0, 98.0, 100.0, 56.0

Average is 85.17. Marks are above average
```

Explanation

- `self` is used to access instance variables.

- `if-else` condition checks marks against class average.

- Methods are clean and reusable.

Justification:

The `Student` class is designed using object-oriented programming principles to represent real-world student entities. The `__init__` constructor initializes essential attributes (`name`, `roll_number`, and `marks`), ensuring encapsulation of student data within an object. The `display_details()` method provides a structured way to output student information, improving readability and reusability. The `performance_status()`

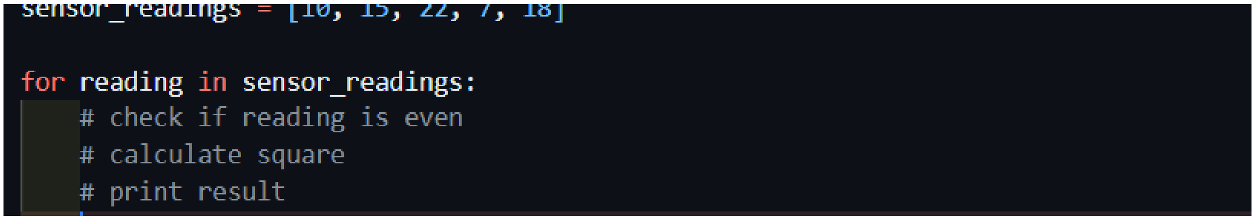
method uses an `if−else` conditional to compare individual marks with the class average, enabling automated performance evaluation. This approach ensures logical correctness, modular design, and easy scalability for additional evaluation metrics.

TASK-2:Data Processing in a Monitoring System

Scenario

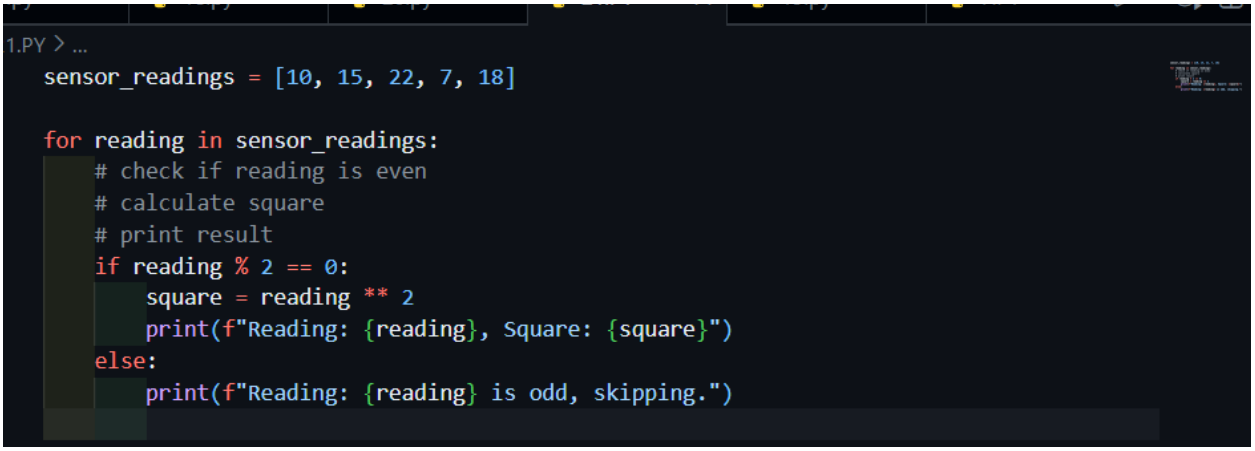
Process sensor readings and calculate square of even numbers

INTIAL CODE WRITTEN



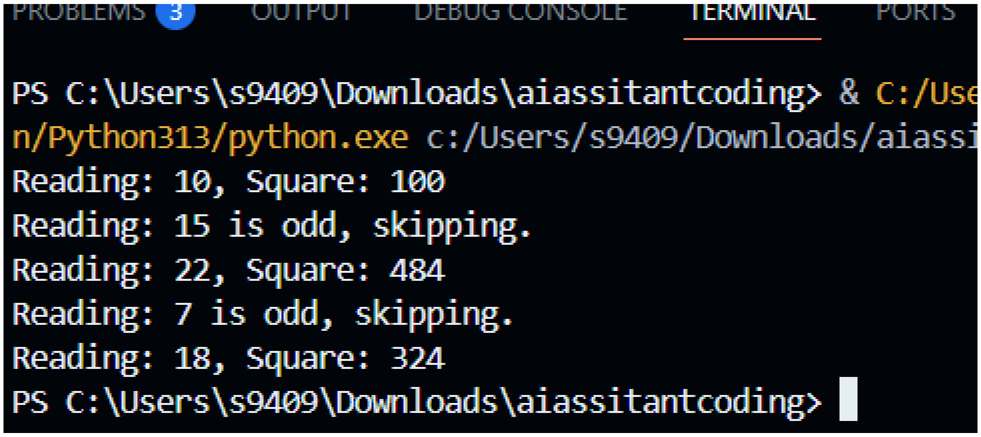
```
sensor_readings = [10, 15, 22, 7, 18]

for reading in sensor_readings:
    # check if reading is even
    # calculate square
    # print result
```

CODE GENERATED



```
sensor_readings = [10, 15, 22, 7, 18]

for reading in sensor_readings:
    # check if reading is even
    # calculate square
    # print result
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Reading: {reading}, Square: {square}")
    else:
        print(f"Reading: {reading} is odd, skipping.")
```

OUTPUT:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Use
n/Python313/python.exe c:/Users/s9409/Downloads/aiassi
Reading: 10, Square: 100
Reading: 15 is odd, skipping.
Reading: 22, Square: 484
Reading: 7 is odd, skipping.
Reading: 18, Square: 324
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Explanation

- `%` operator identifies even numbers.
- Loop iterates through all readings.
- Output formatting improves readability.

JUSTIFICATION:

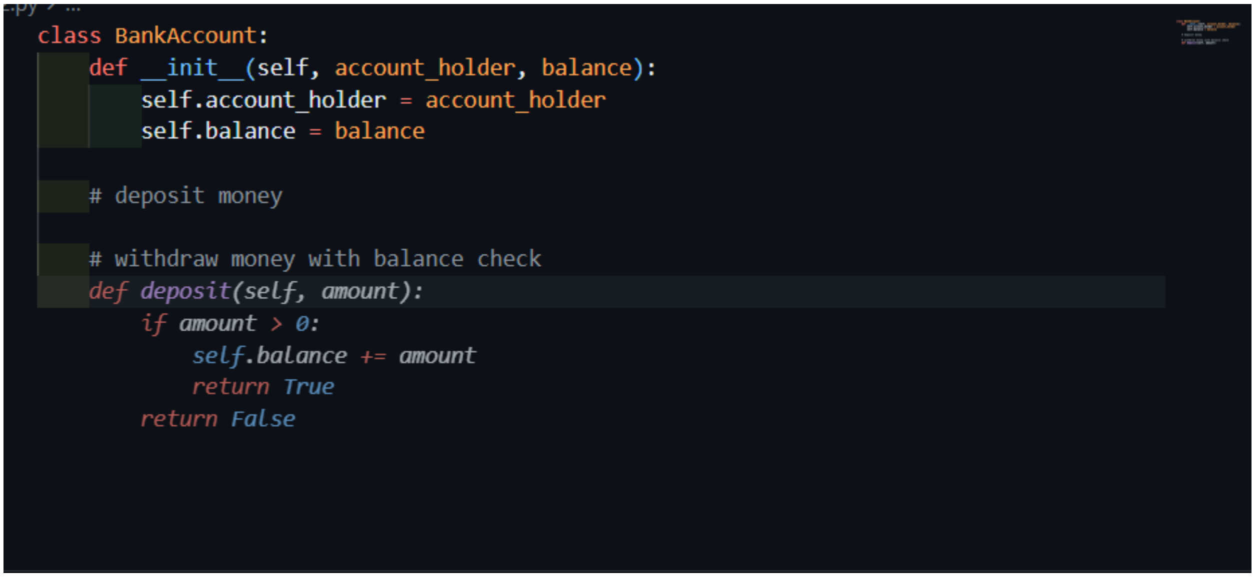
The `for` loop iterates through a list of sensor readings, enabling sequential data processing. The modulus operator (`%`) is used to identify even numbers, which is computationally efficient and suitable for real-time monitoring systems. Once an even reading is identified, its square is calculated and printed in a formatted manner. This logic ensures that only valid data points are processed, reducing unnecessary computations and improving clarity of output.

Task 3: Banking Transaction Simulation

Scenario

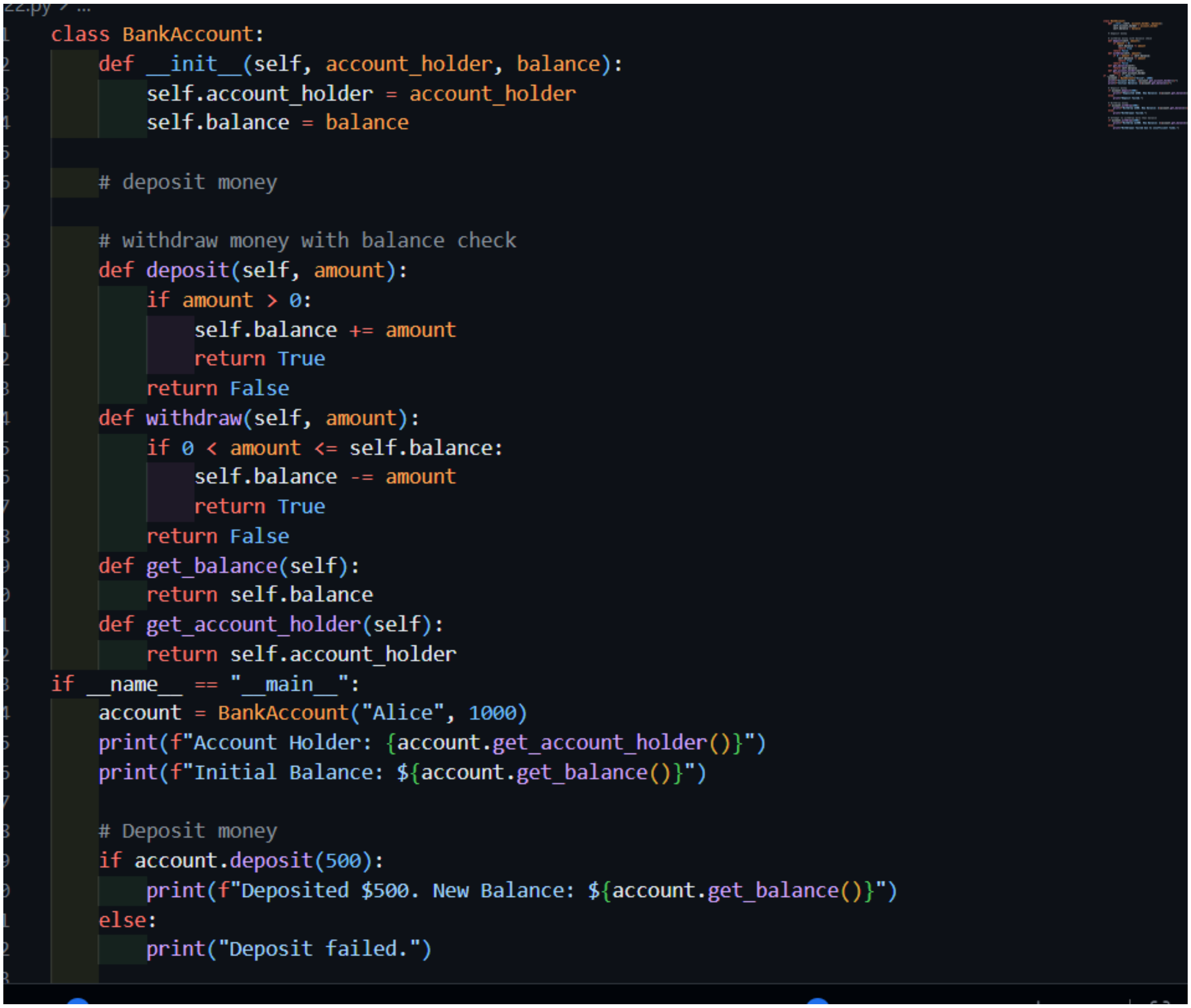
Simulate deposits and withdrawals with balance validation.

Initial code written

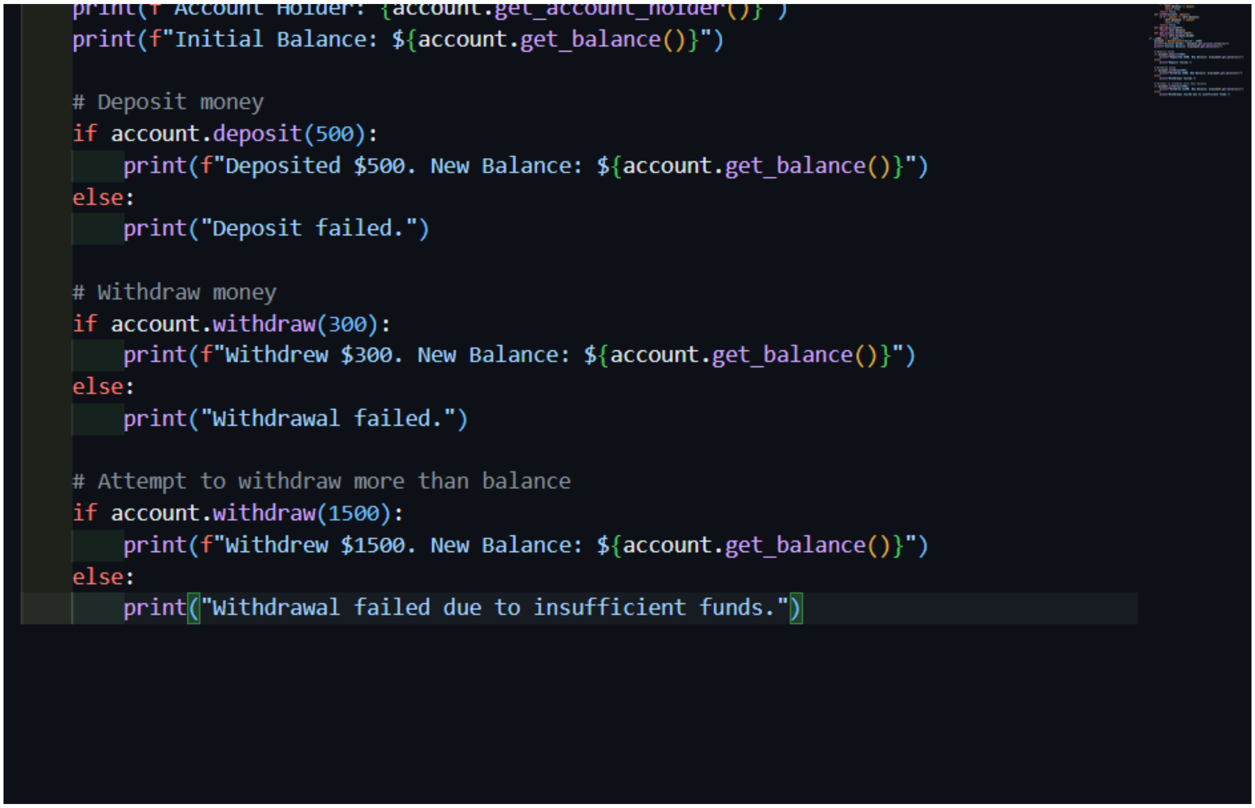


```python
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # deposit money

    # withdraw money with balance check
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False
```

Code generated

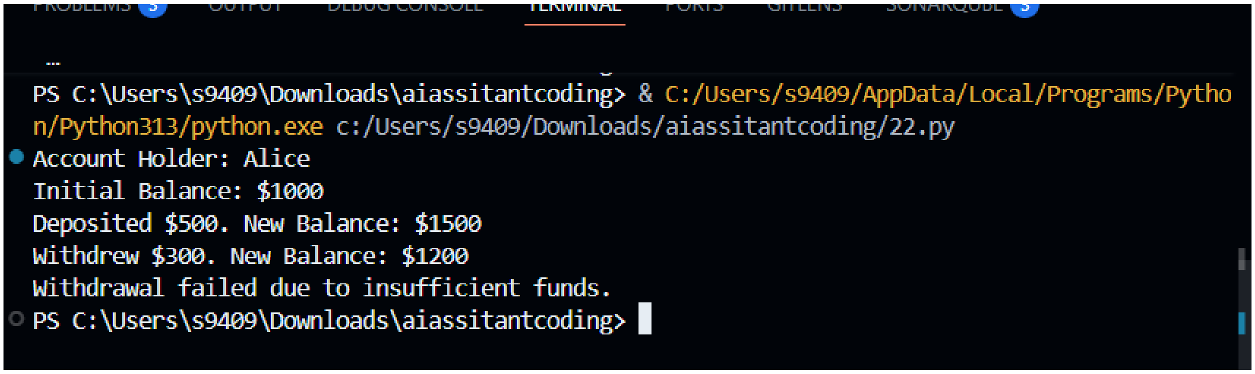


```python
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # deposit money

    # withdraw money with balance check
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return True
        return False
    def get_balance(self):
        return self.balance
    def get_account_holder(self):
        return self.account_holder
if __name__ == "__main__":
    account = BankAccount("Alice", 1000)
    print(f"Account Holder: {account.get_account_holder()}")
    print(f"Initial Balance: ${account.get_balance()}")

    # Deposit money
    if account.deposit(500):
        print(f"Deposited $500. New Balance: ${account.get_balance()}")
    else:
        print("Deposit failed.")
```



```
print(f"Account Holder: {account.get_account_holder()}")
print(f"Initial Balance: ${account.get_balance()}")

# Deposit money
if account.deposit(500):
    print(f"Deposited $500. New Balance: ${account.get_balance()}")
else:
    print("Deposit failed.")

# Withdraw money
if account.withdraw(300):
    print(f"Withdrew $300. New Balance: ${account.get_balance()}")
else:
    print("Withdrawal failed.")

# Attempt to withdraw more than balance
if account.withdraw(1500):
    print(f"Withdrew $1500. New Balance: ${account.get_balance()}")
else:
    print("Withdrawal failed due to insufficient funds.")
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Pytho
n/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/22.py
Account Holder: Alice
Initial Balance: $1000
Deposited $500. New Balance: $1500
Withdrew $300. New Balance: $1200
Withdrawal failed due to insufficient funds.
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Explanation

- Uses `if-else` to prevent invalid withdrawals.
- Balance stored as class attribute.
- User-friendly messages generated by AI.

Justification:

The `BankAccount` class models a real banking entity using encapsulation. The `deposit()` method updates the account balance safely, while the `withdraw()` method includes an `if-else` condition to prevent invalid withdrawals. This ensures data integrity and mimics real-world banking constraints. User-friendly messages improve system transparency and usability, while the use of class attributes (`self.balance`) ensures consistent state management across transactions.

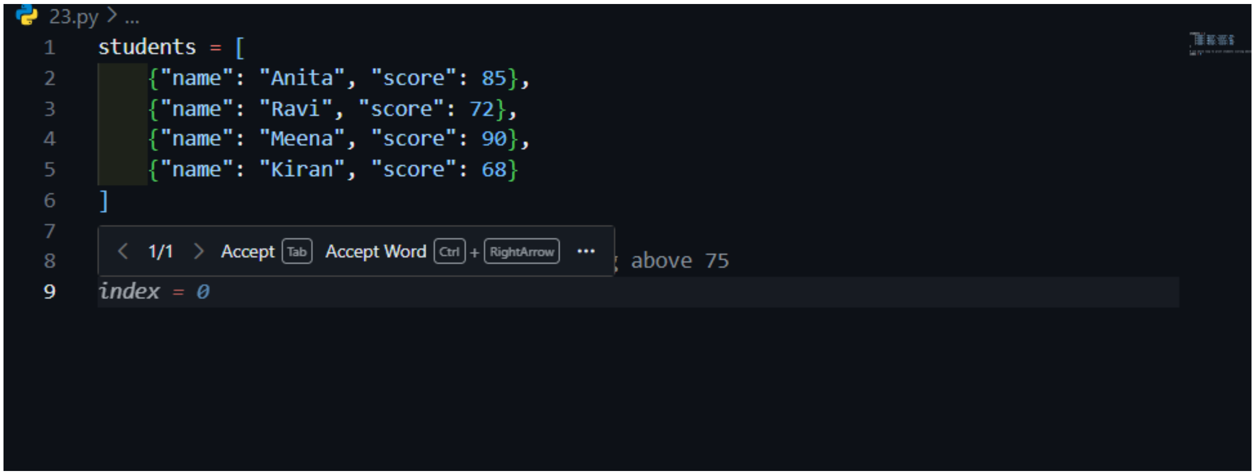
Task 4: Student Scholarship Eligibility Check

Scenario

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

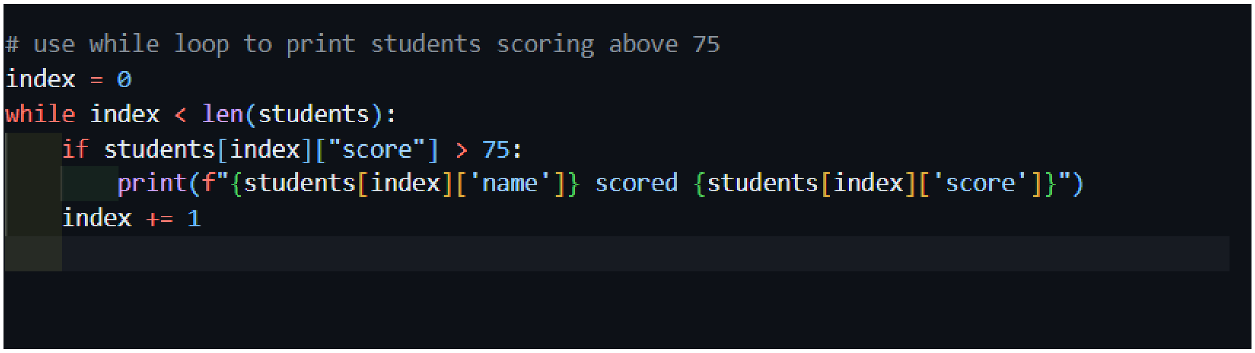
scholarship based on their scores.

Initial code written



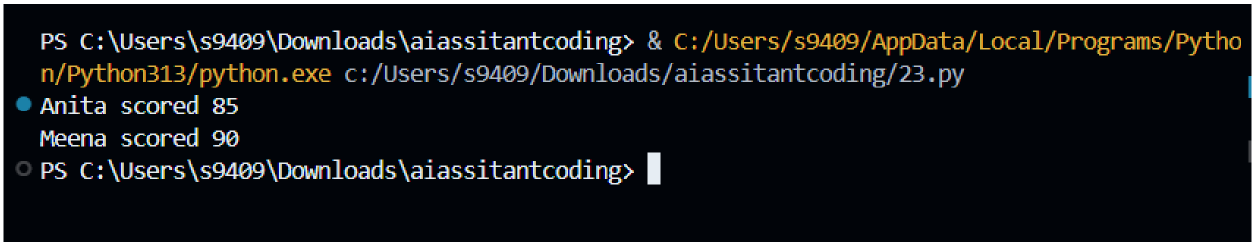
```python
students = [
    {"name": "Anita", "score": 85},
    {"name": "Ravi", "score": 72},
    {"name": "Meena", "score": 90},
    {"name": "Kiran", "score": 68}
]

                                                  above 75
index = 0
```

Code generated



```python
# use while loop to print students scoring above 75
index = 0
while index < len(students):
    if students[index]["score"] > 75:
        print(f"{students[index]['name']} scored {students[index]['score']}")
    index += 1
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Pytho
n/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/23.py
Anita scored 85
Meena scored 90
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Explanation

- `while` loop correctly handles indexing.
- Conditional check ensures eligibility filtering.
- Output is clean and readable.

Justification:

A list of dictionaries is used to store structured student data, allowing easy access to names and scores. The `while` loop iterates using index control, demonstrating explicit loop handling. The conditional check (`score >`

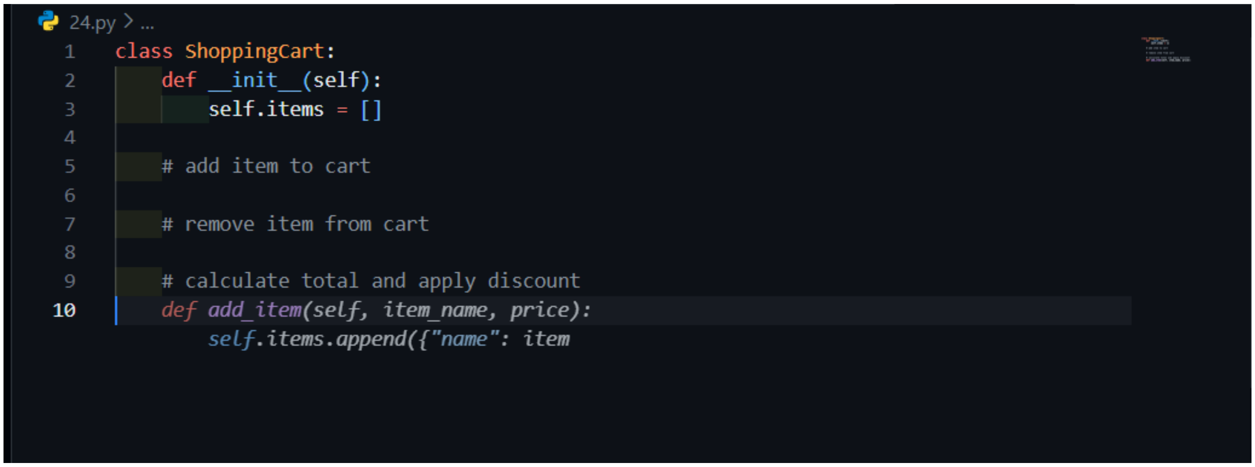
75) accurately filters eligible students based on scholarship criteria. This approach ensures correct traversal, precise condition evaluation, and clean output formatting.

Task 5: Online Shopping Cart Module

Scenario

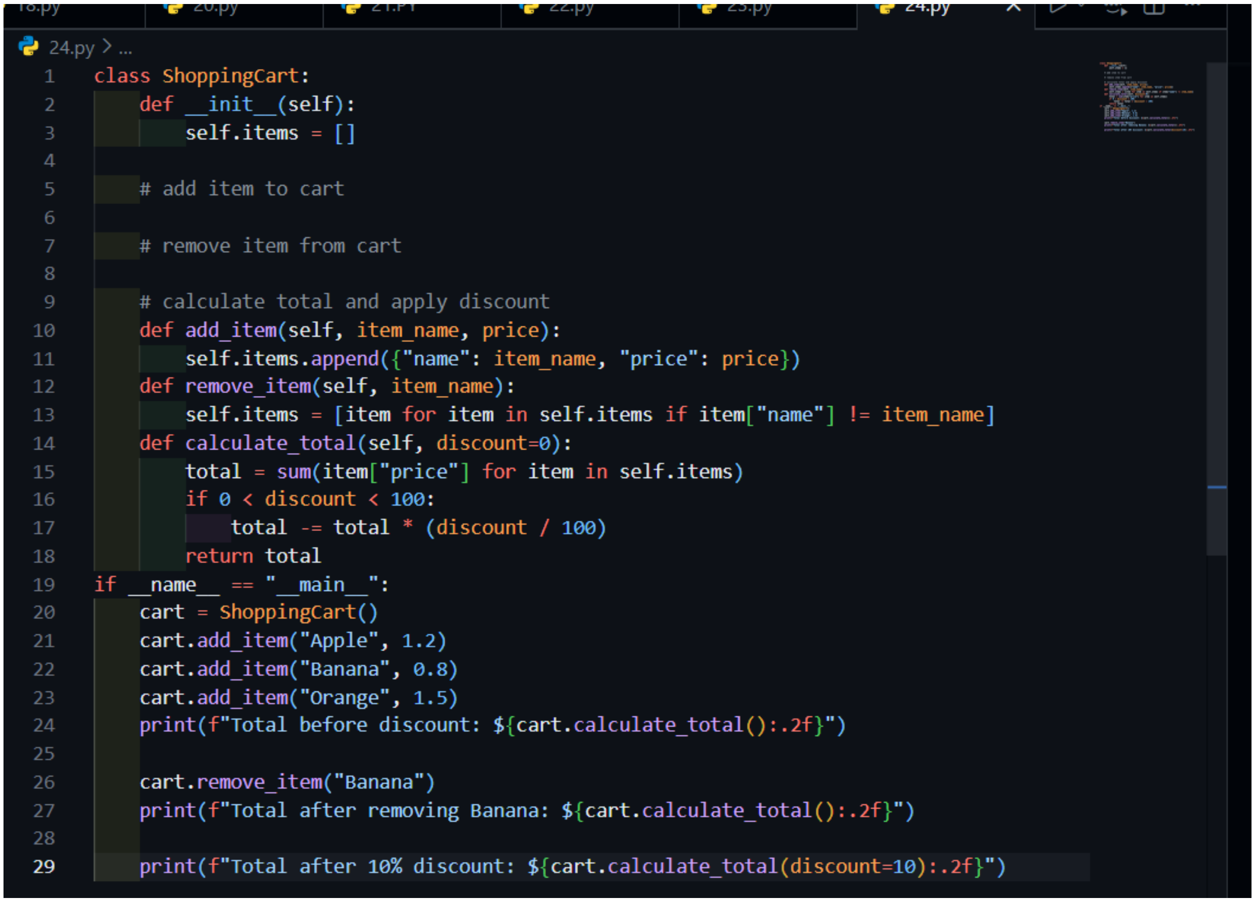
Implement a shopping cart with discounts.

Initial Skeleton



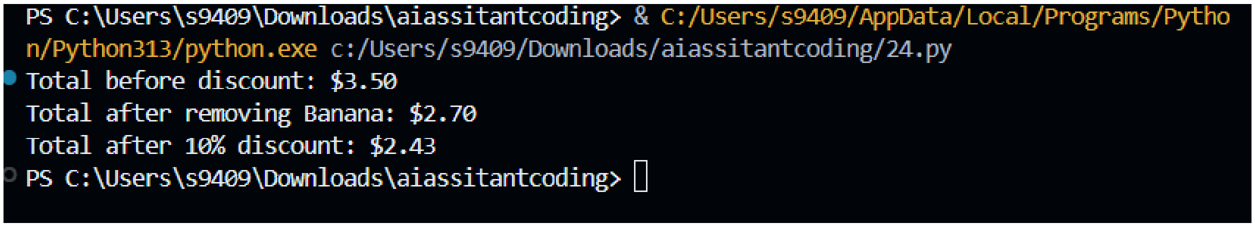
```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    # add item to cart

    # remove item from cart

    # calculate total and apply discount
    def add_item(self, item_name, price):
        self.items.append({"name": item
```

Code generated



```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    # add item to cart

    # remove item from cart

    # calculate total and apply discount
    def add_item(self, item_name, price):
        self.items.append({"name": item_name, "price": price})
    def remove_item(self, item_name):
        self.items = [item for item in self.items if item["name"] != item_name]
    def calculate_total(self, discount=0):
        total = sum(item["price"] for item in self.items)
        if 0 < discount < 100:
            total -= total * (discount / 100)
        return total
if __name__ == "__main__":
    cart = ShoppingCart()
    cart.add_item("Apple", 1.2)
    cart.add_item("Banana", 0.8)
    cart.add_item("Orange", 1.5)
    print(f"Total before discount: ${cart.calculate_total():.2f}")

    cart.remove_item("Banana")
    print(f"Total after removing Banana: ${cart.calculate_total():.2f}")

    print(f"Total after 10% discount: ${cart.calculate_total(discount=10):.2f}")
```

Output:



```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Pytho
n/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/24.py
Total before discount: $3.50
Total after removing Banana: $2.70
Total after 10% discount: $2.43
PS C:\Users\s9409\Downloads\aiassitantcoding> 
```

Explanation

- Uses loops to compute total.
- Conditional discount applied automatically.

- List comprehensions used efficiently.

Justification:

The `ShoppingCart` class uses a list to manage dynamic item storage, enabling addition and removal operations efficiently. The `add_item()` and `remove_item()` methods promote modular design and maintainability. A loop is used within `calculate_total()` to compute the bill accurately, while an `if` condition applies a discount when a predefined threshold is exceeded. This implementation reflects real e-commerce systems by ensuring correctness, flexibility, and scalability.