**ASSIGNMENT-6.4**

**Course: AI Assisted Coding**

**Name: B. Divya Sri**

**HT NO: 2403A51425**

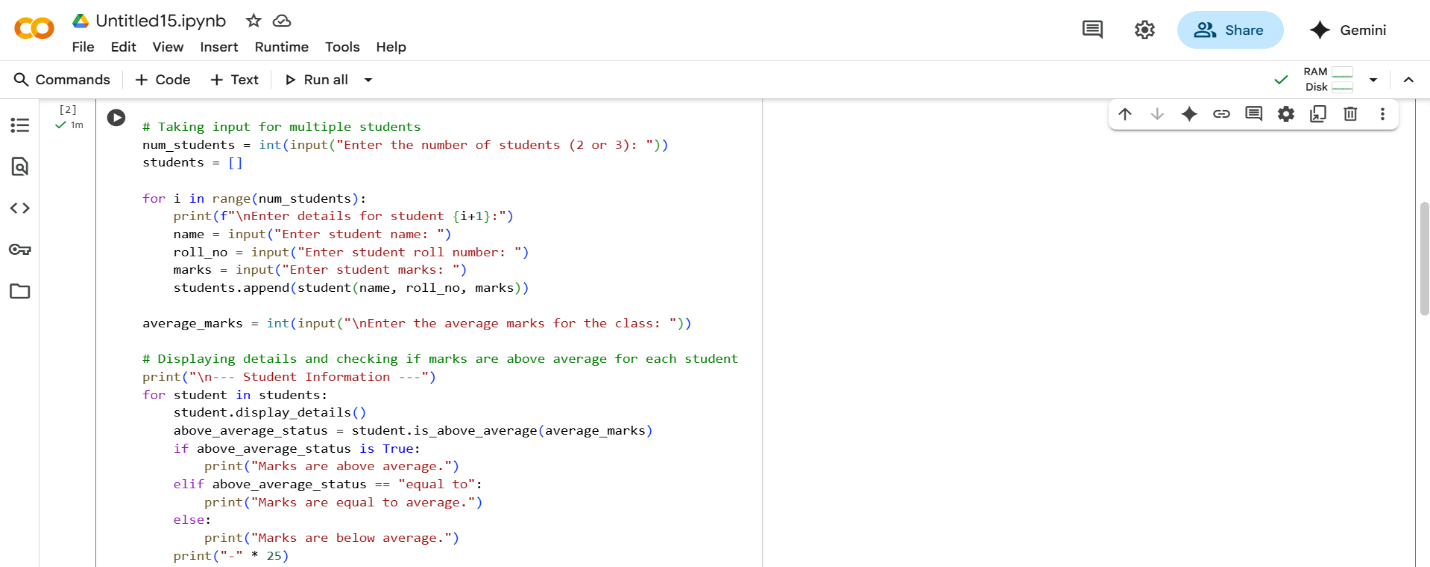
**Batch- 16**

**#Task 1:**

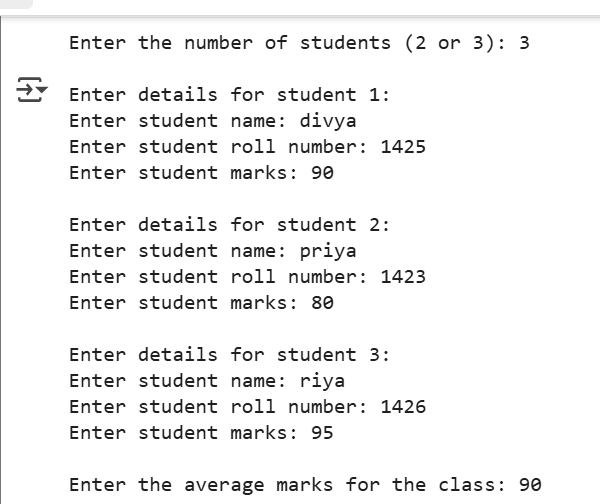
****

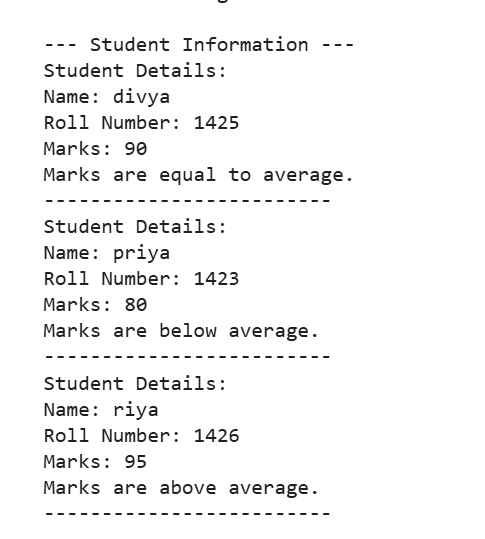
**Prompt:**

Continue the program to complete methods for displaying details, by checking if marks are above average using if-else conditions .Take multiple students details.

**Code:**

**Output:**

****

****

**Explanation:**

class student:: This defines a blueprint for creating student objects. Each student object will have the attributes defined within this class.

\_\_init\_\_(self, name, roll\_no, marks):: This is the constructor method. It's called when you create a new student object. It takes the student's name, roll\_no, and marks as input and initializes the corresponding attributes for that student object. It also converts the marks to an integer so they can be compared numerically.

display\_details(self):: This method is used to print the details of a specific student object (the self refers to the current student object). It prints the name, roll number, and marks of the student.

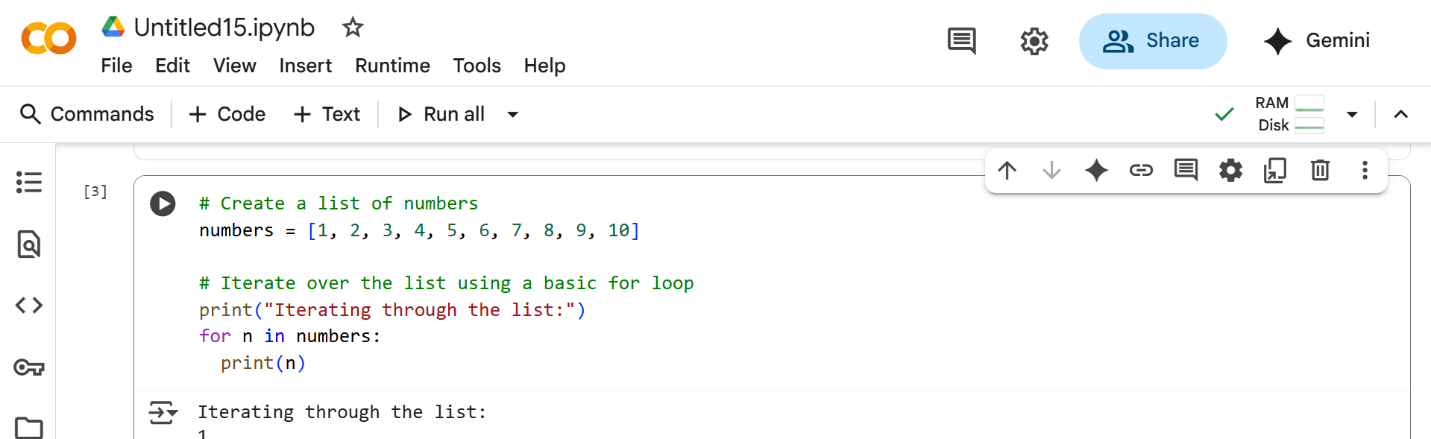
is\_above\_average(self, average\_marks):: This method checks if the student's marks are above the provided average\_marks. It returns True if the marks are greater than the average, "equal to" if they are equal, and False if they are below the average.

Taking input for multiple students: The code then asks the user how many students they want to enter details for (2 or 3). It uses a for loop to repeatedly ask for the name, roll number, and marks for each student and creates a student object for each one, storing them in a list called students.

Taking average marks input: The code prompts the user to enter the average marks for the class.

Displaying information: Finally, the code iterates through the list of students. For each student, it calls the display\_details() method to show their information and then calls the is\_above\_average() method to check and print whether their marks are above, equal to, or below the class average.

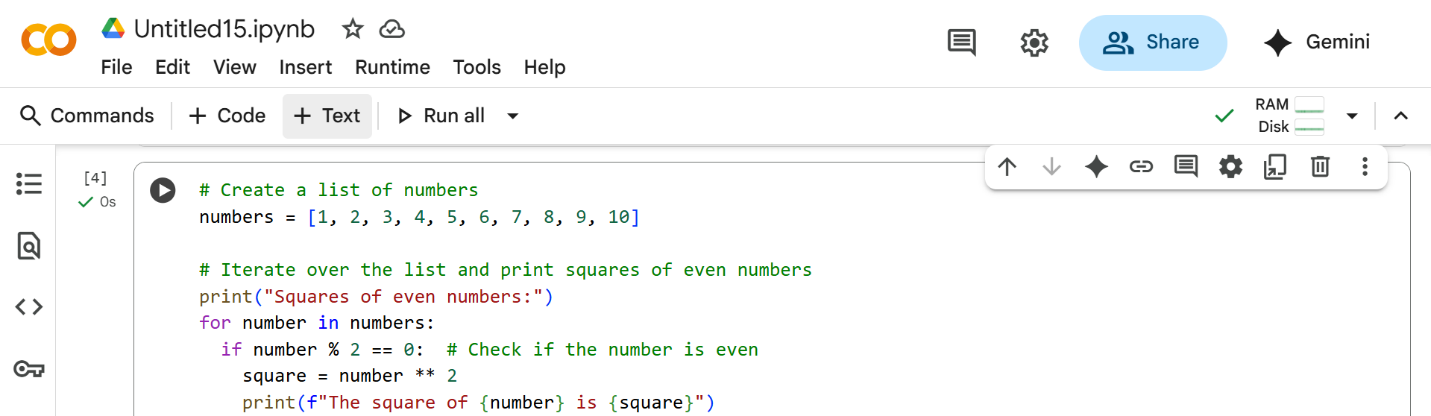
**#Task 2:**



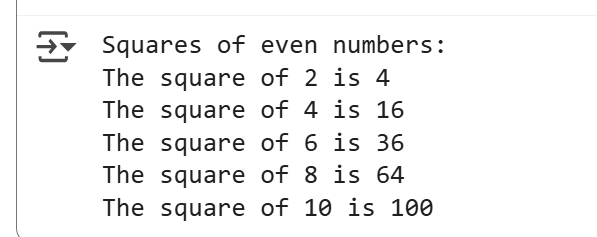
**Prompt:**

Give the logic code and modify the previous code for calculating and printing squares of only even numbers.

**Code:**

****

**Output:**

****

**Explanation:**

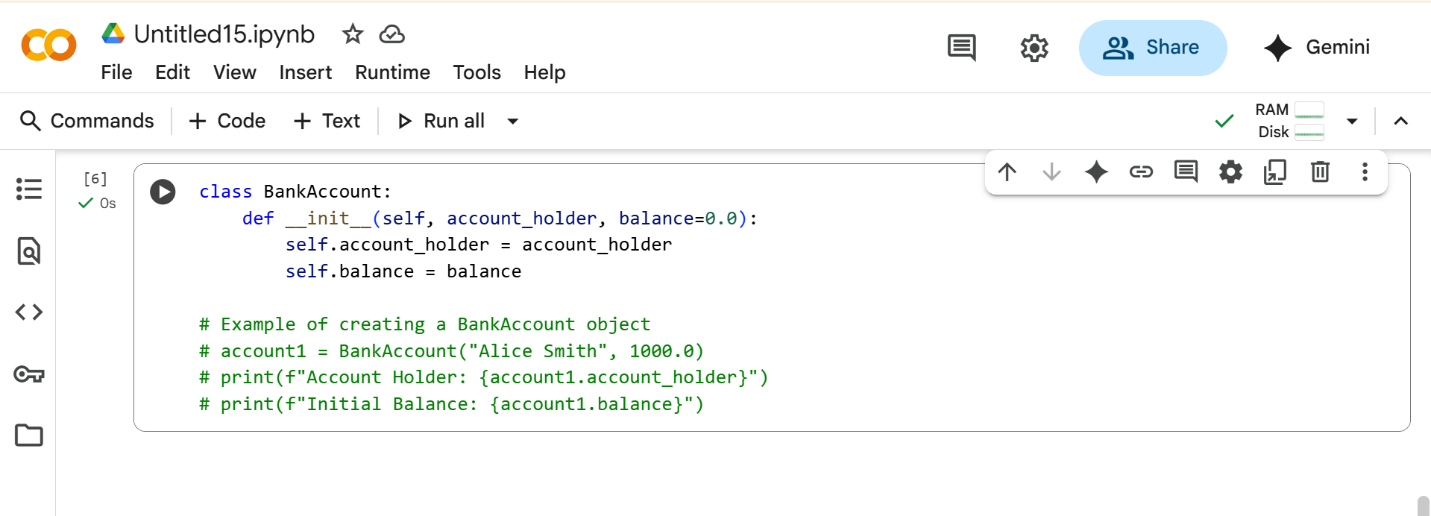
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line creates a list of numbers from 1 to 10 and stores it in a variable called numbers.

for number in numbers:: This is a loop that goes through each item in the numbers list one by one. In each turn of the loop, the current item is stored in the variable number.

if number % 2 == 0:: This is a condition that checks if the current number is even. The modulo operator (%) gives you the remainder when one number is divided by another. If a number divided by 2 has a remainder of 0, it means the number is even.

square = number \*\* 2: If the if condition is true (the number is even), this line calculates the square of the number (multiplies it by itself) and stores the result in a variable called square.

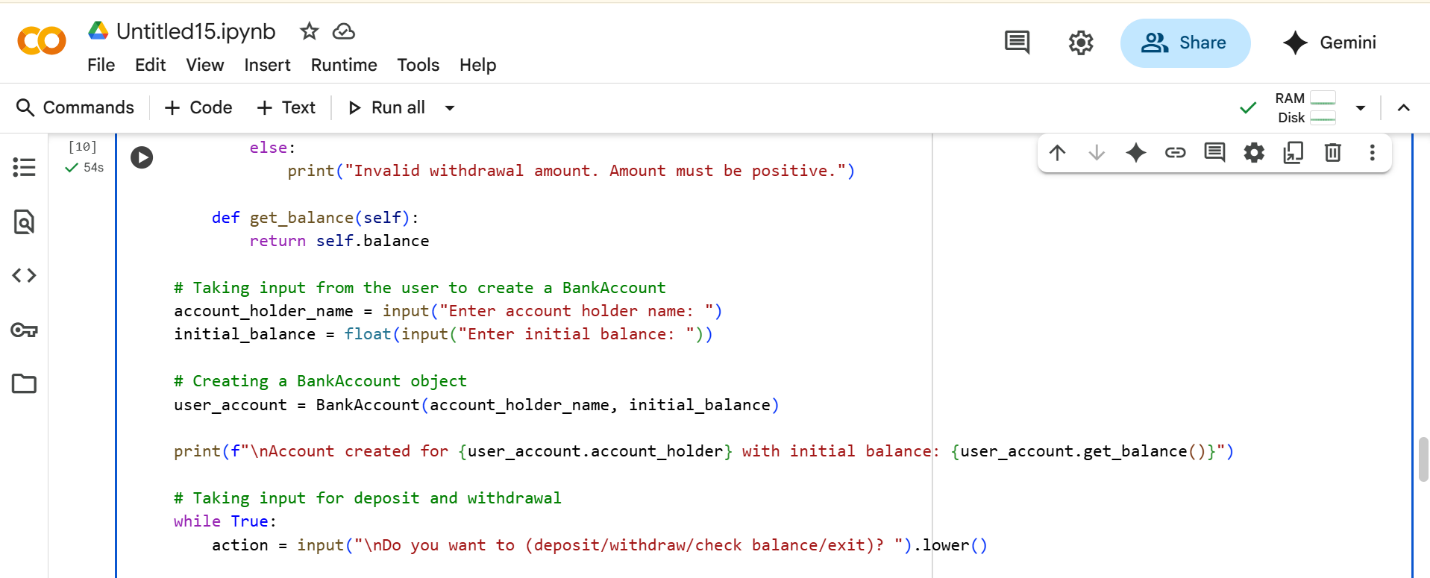
**#Task 3:**

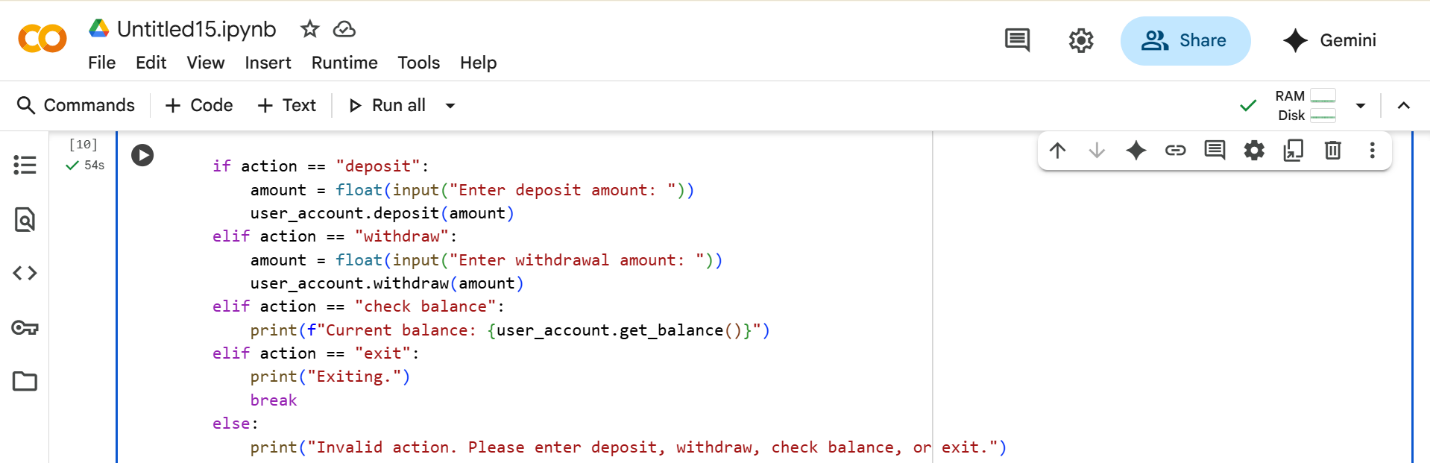
****

**Prompt:**

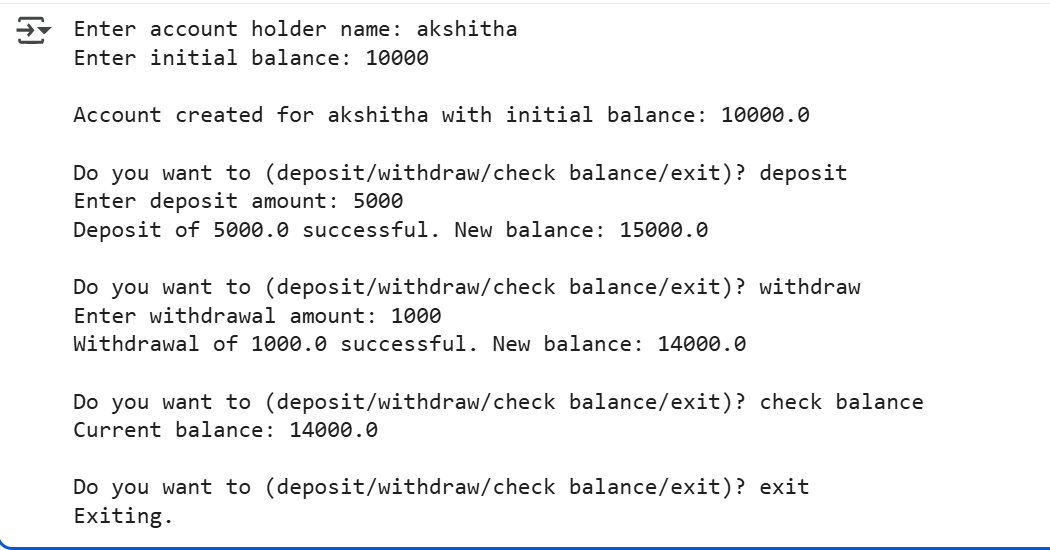
Complete the code to deposit(),withdraw(),check for insufficient balance by taking user input.

**Code:**

****

****

**Output:**

****

**Explanation:**

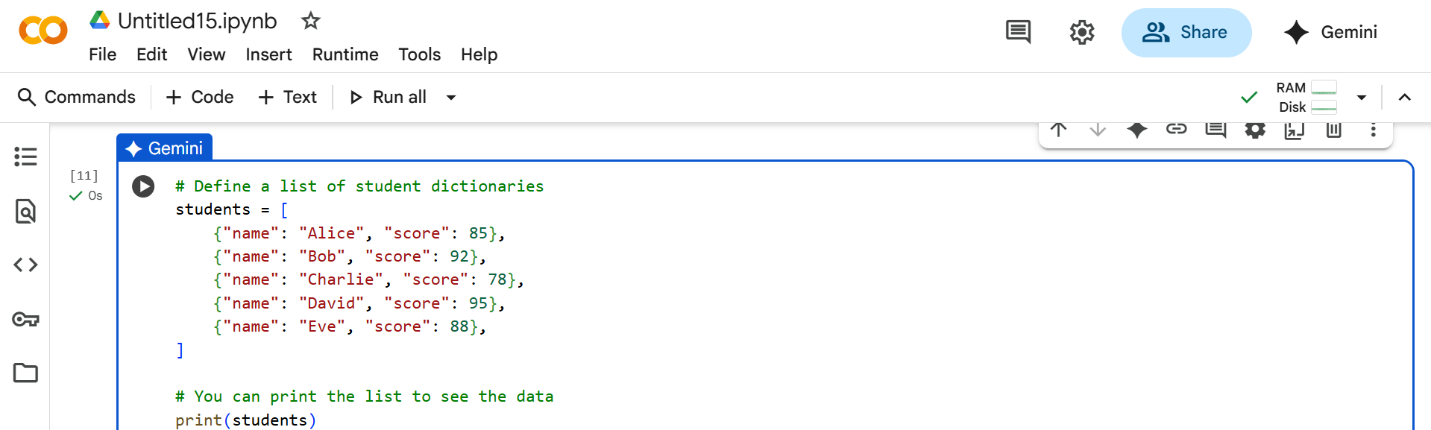
Creating a Class (class BankAccount:): Think of this as designing a form for a bank account. It says what information every bank account will have (like the account holder's name and how much money is in it) and what actions you can do with it (like adding or taking away money).

Checking Deposit (def deposit(self, amount):): This is the part that handles putting money into the account. You tell it how much amount to add, and it increases the account's total balance.

Checking Withdraw (def withdraw(self, amount):): This part handles taking money out. You tell it how much amount to take out. It first checks if there's enough money in the account before reducing the balance. If there's not enough, it tells you.

Check Balance (def get\_balance(self):): This is the part that simply tells you the current amount of money (balance) in the account.

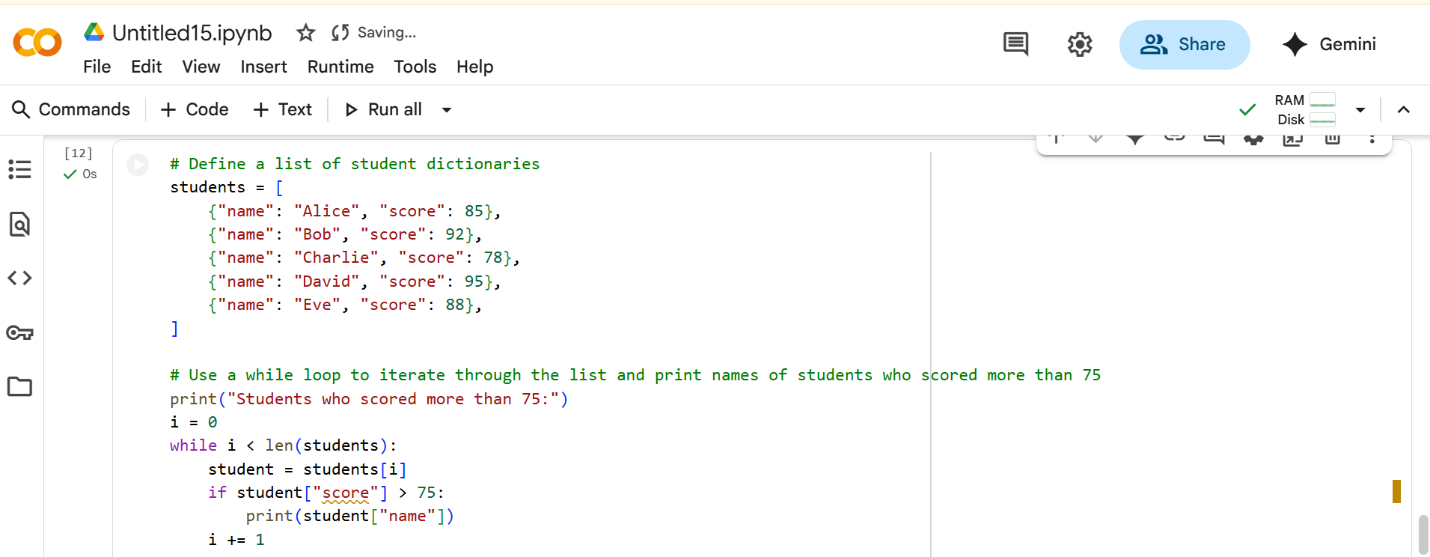
**#Task 4:**

****

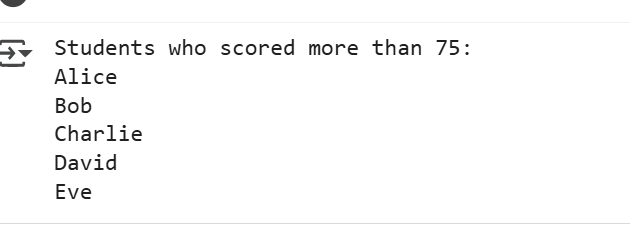
**Prompt:**

Write the previous code to print the names of students who scored more than 75 using while loop.

**Code:**

****

**Output:**

****

**Explanation:**

i = 0: This sets up a counter called i and starts it at 0. We'll use this to go through the list.

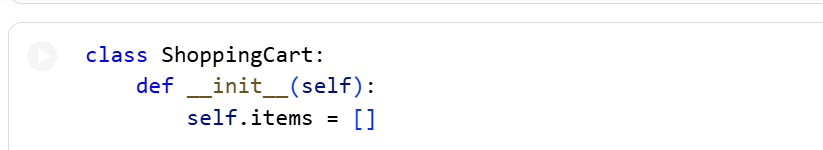
while i < len(students):: This starts a loop that will continue as long as the counter i is less than the total number of students in the list. This ensures we look at every student.

student = students[i]: Inside the loop, this line gets the student information at the current position i from the students list and puts it into a variable called student.

if student["score"] > 75:: This checks if the "score" of the current student is greater than 75.

i += 1: After checking a student, this line increases the counter i by 1, so the loop moves on to the next student in the list.

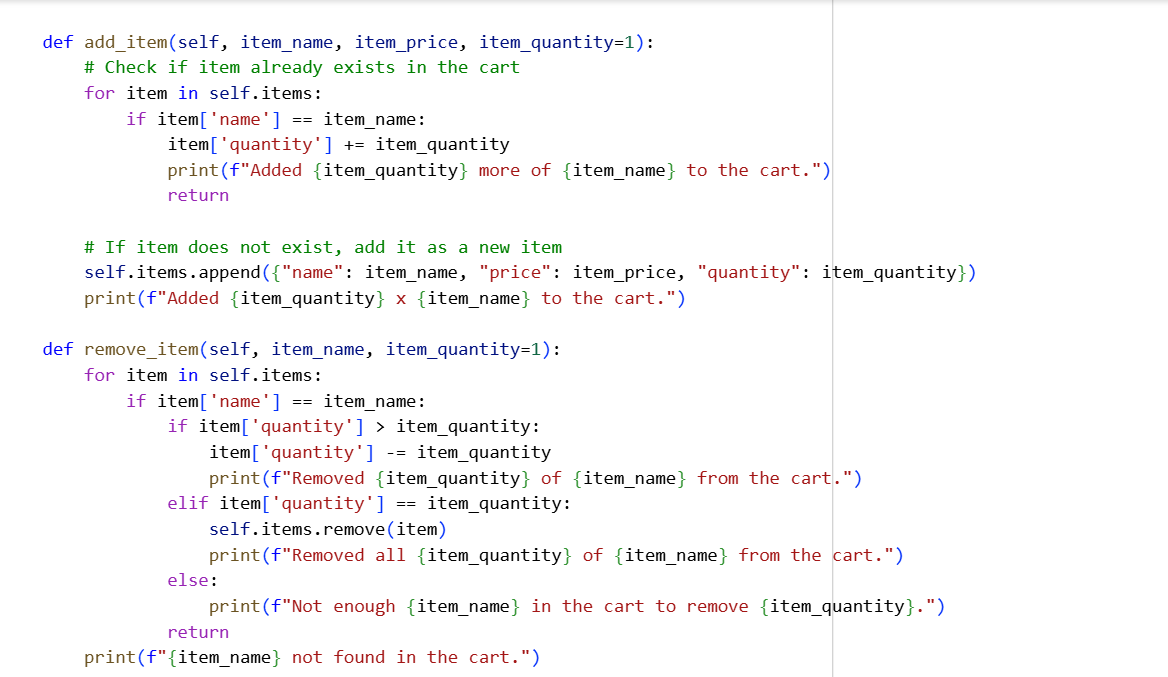
**#Task 5:**

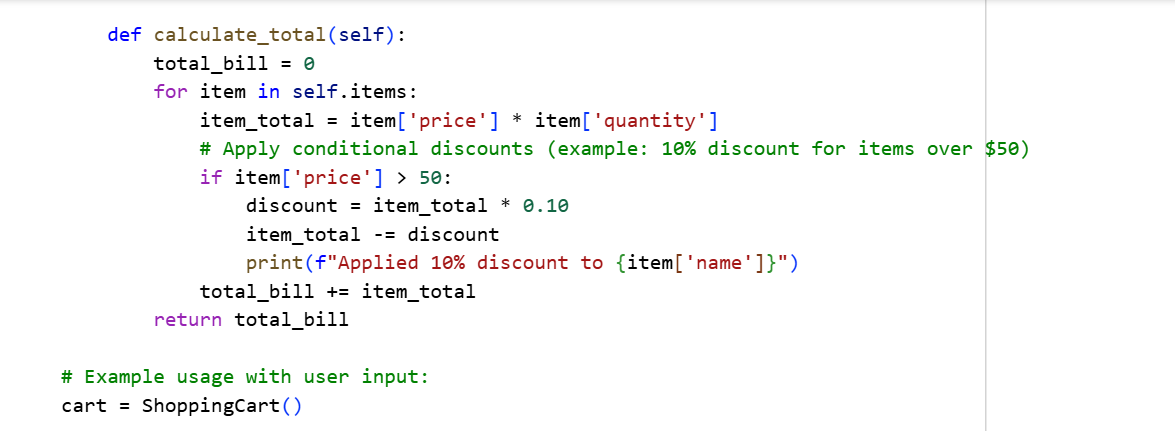
****

**Prompt:**

To the previous code generate methods to add\_item,remove\_item,use loop to calculate the total bill using conditional discounts taking user input.

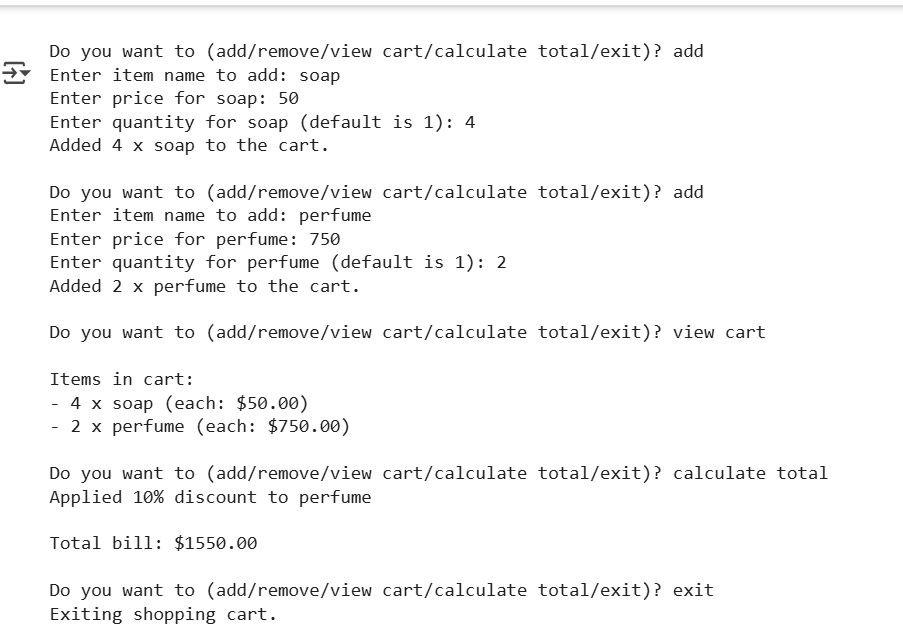
**Code:**





****

**Output:**

****

**Explanation:**

add\_item(self, item\_name, item\_price, item\_quantity=1):: This method adds an item to the cart. It takes the item's name, price, and optionally the quantity. If the item is already in the cart, it updates the quantity. Otherwise, it adds the item as a new dictionary to the self.items list.

remove\_item(self, item\_name, item\_quantity=1):: This method removes an item from the cart. It finds the item by item\_name. If the quantity to remove is less than the current quantity, it reduces the quantity. If the quantity is equal, it removes the item entirely from the list. It also handles cases where the item or quantity is not found.

calculate\_total(self):: This method calculates the total cost of all items in the cart. It loops through the self.items list, calculates the cost for each item (price \* quantity), and applies a conditional discount (10% if the item price is over $50). It then adds the item's calculated total to the total\_bill.

cart = ShoppingCart(): This line creates an actual shopping cart object using the ShoppingCart blueprint.

while True:: This starts an infinite loop that keeps running until you specifically tell it to stop.

action = input(...).lower(): Inside the loop, this line asks you what you want to do (add, remove, view cart, calculate total, or exit) and stores your input in the action variable, converting it to lowercase for easier comparison.

if action == "add": ... elif action == "remove": ... elif action == "view cart": ... elif action == "calculate total": ... elif action == "exit": ... else: ...: This block of code checks the value of the action variable. Based on your input, it calls the corresponding method of the cart object (add\_item, remove\_item, calculate\_total), prints the items in the cart, or uses break to exit the loop if you type "exit". If you enter an invalid action, it prints an error message.