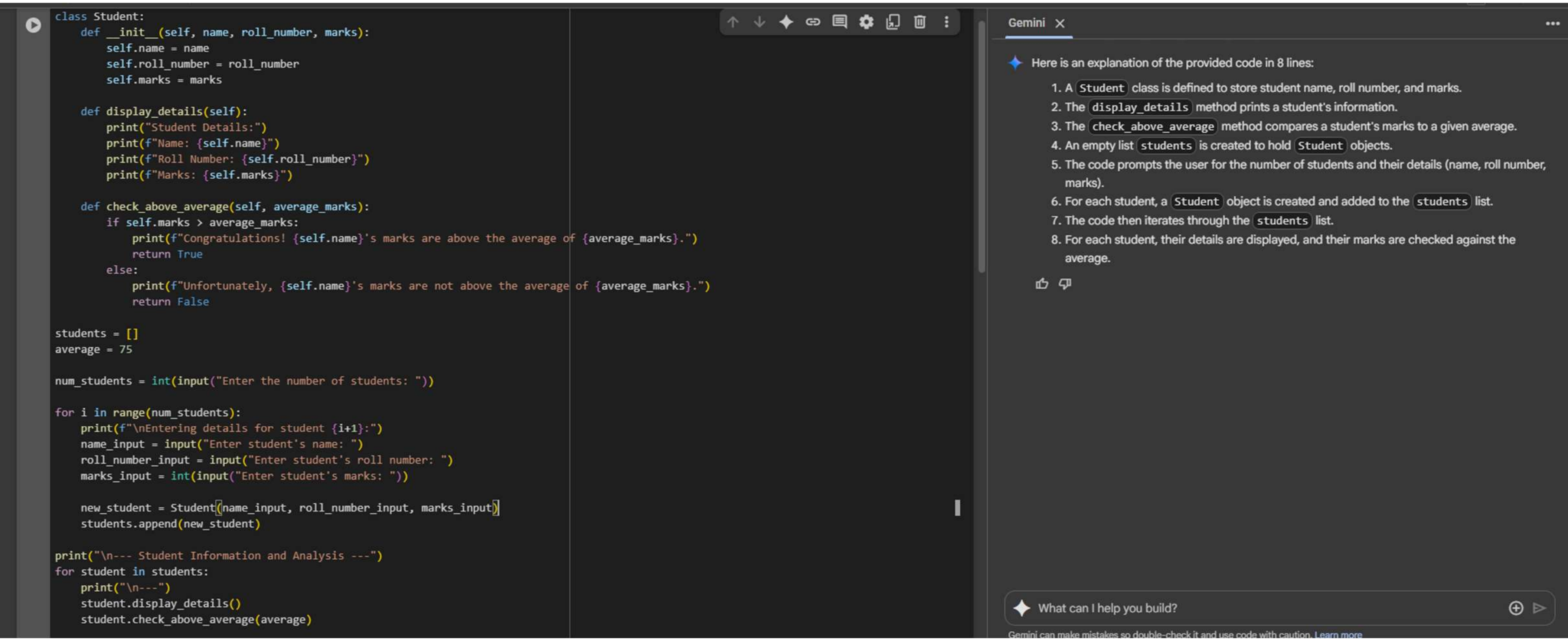
# Assignment-6

## Task-1:

Prompt: write a python program to create a class named student with attribute name,roll_number,and marks for displaying details and checking the marks above are average give the code in the way so we can give name and roll number and marks to check



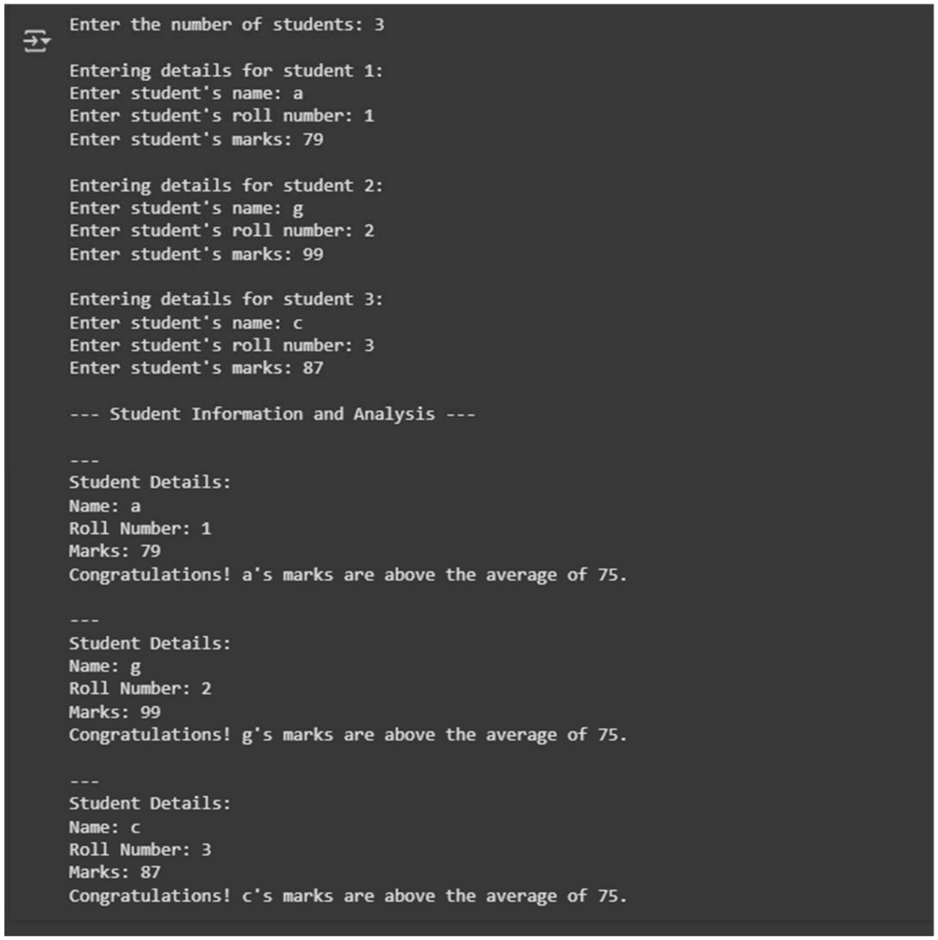
```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print("Student Details:")
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def check_above_average(self, average_marks):
        if self.marks > average_marks:
            print(f"Congratulations! {self.name}'s marks are above the average of {average_marks}.")
            return True
        else:
            print(f"Unfortunately, {self.name}'s marks are not above the average of {average_marks}.")
            return False

students = []
average = 75

num_students = int(input("Enter the number of students: "))

for i in range(num_students):
    print(f"\nEntering details for student {i+1}:")
    name_input = input("Enter student's name: ")
    roll_number_input = input("Enter student's roll number: ")
    marks_input = int(input("Enter student's marks: "))

    new_student = Student(name_input, roll_number_input, marks_input)
    students.append(new_student)

print("\n--- Student Information and Analysis ---")
for student in students:
    print("\n---")
    student.display_details()
    student.check_above_average(average)
```

Here is an explanation of the provided code in 8 lines:

1. A `Student` class is defined to store student name, roll number, and marks.
2. The `display_details` method prints a student's information.
3. The `check_above_average` method compares a student's marks to a given average.
4. An empty list `students` is created to hold `Student` objects.
5. The code prompts the user for the number of students and their details (name, roll number, marks).
6. For each student, a `Student` object is created and added to the `students` list.
7. The code then iterates through the `students` list.
8. For each student, their details are displayed, and their marks are checked against the average.

Gemini can make mistakes so double-check it and use code with caution. Learn more

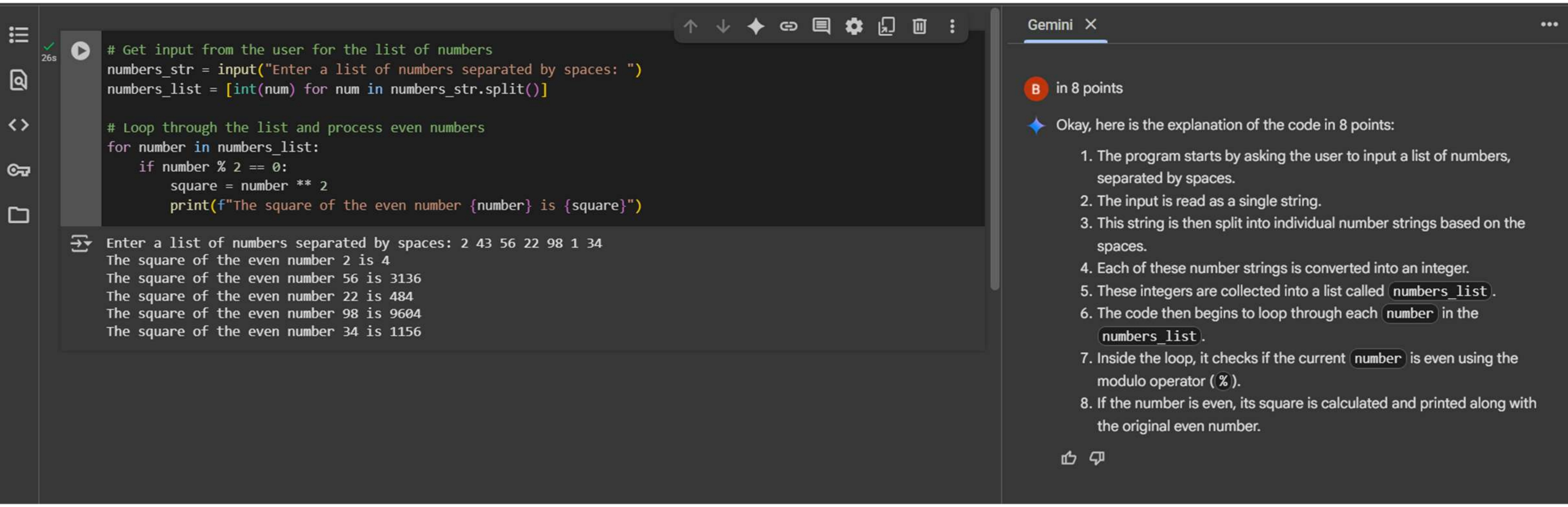
Output:



```
Enter the number of students: 3

Entering details for student 1:
Enter student's name: a
Enter student's roll number: 1
Enter student's marks: 79

Entering details for student 2:
Enter student's name: g
Enter student's roll number: 2
Enter student's marks: 99

Entering details for student 3:
Enter student's name: c
Enter student's roll number: 3
Enter student's marks: 87

--- Student Information and Analysis ---

---
Student Details:
Name: a
Roll Number: 1
Marks: 79
Congratulations! a's marks are above the average of 75.

---
Student Details:
Name: g
Roll Number: 2
Marks: 99
Congratulations! g's marks are above the average of 75.

---
Student Details:
Name: c
Roll Number: 3
Marks: 87
Congratulations! c's marks are above the average of 75.
```

Task-2:

Prompt:write a pythone code pf first two lines of a loop to iterate through a list of numbers and calculate and print the square of even numbers only as dynamic inputs by user for list



```
# Get input from the user for the list of numbers
numbers_str = input("Enter a list of numbers separated by spaces: ")
numbers_list = [int(num) for num in numbers_str.split()]

# Loop through the list and process even numbers
for number in numbers_list:
    if number % 2 == 0:
        square = number ** 2
        print(f"The square of the even number {number} is {square}")
```

```
Enter a list of numbers separated by spaces: 2 43 56 22 98 1 34
The square of the even number 2 is 4
The square of the even number 56 is 3136
The square of the even number 22 is 484
The square of the even number 98 is 9604
The square of the even number 34 is 1156
```

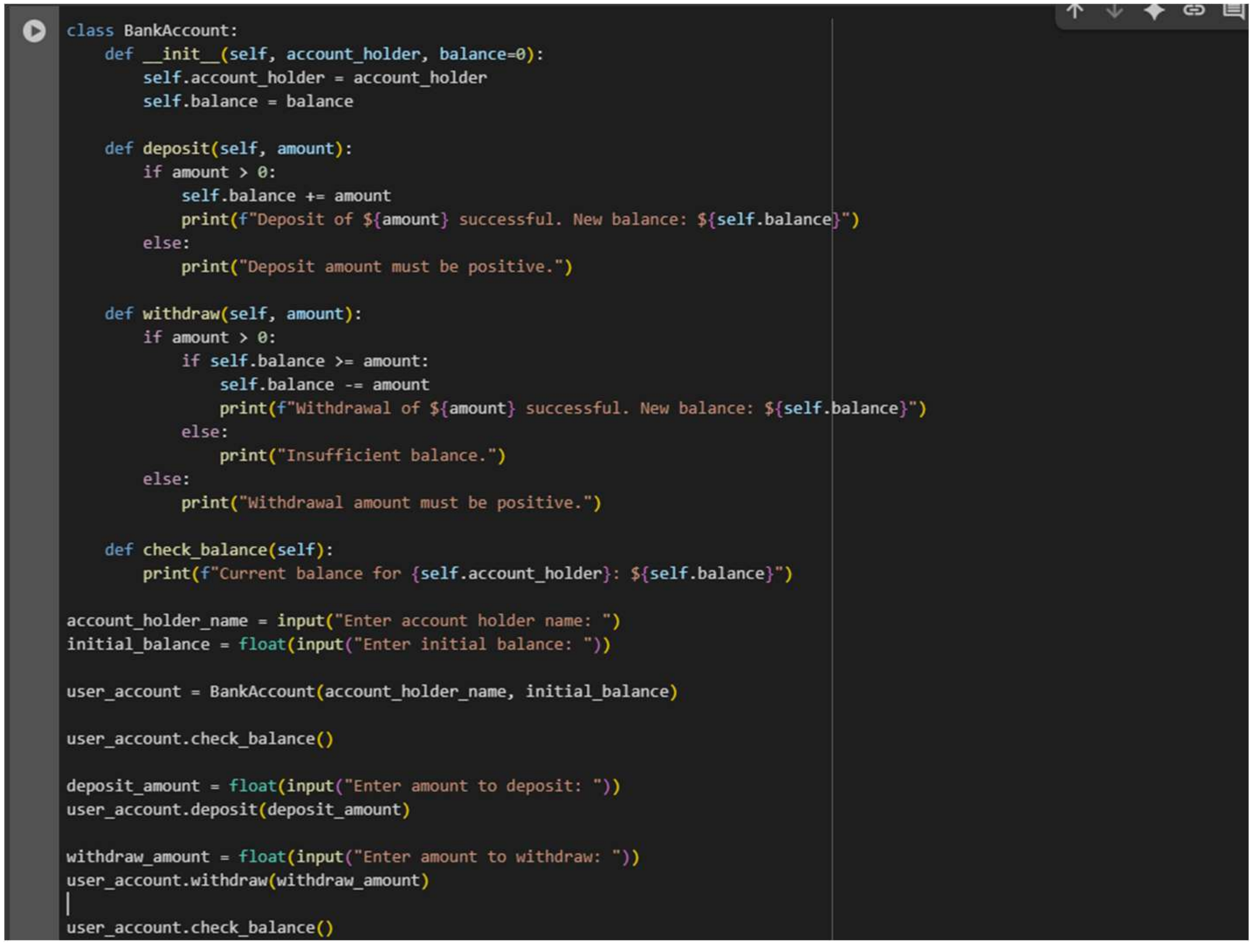
B in 8 points

Okay, here is the explanation of the code in 8 points:

1. The program starts by asking the user to input a list of numbers, separated by spaces.
2. The input is read as a single string.
3. This string is then split into individual number strings based on the spaces.
4. Each of these number strings is converted into an integer.
5. These integers are collected into a list called `numbers_list`.
6. The code then begins to loop through each `number` in the `numbers_list`.
7. Inside the loop, it checks if the current `number` is even using the modulo operator (`%`).
8. If the number is even, its square is calculated and printed along with the original even number.

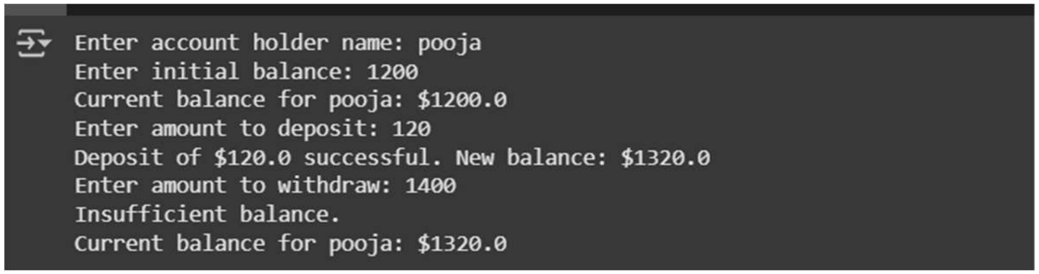
Task-3:

Prompt:write a python program for by creating a class called bankAccount with attributes account_holder and balance and complete method for deposits(),withdraw(),and check for insufficient balance by giver the dynamic inputs by user



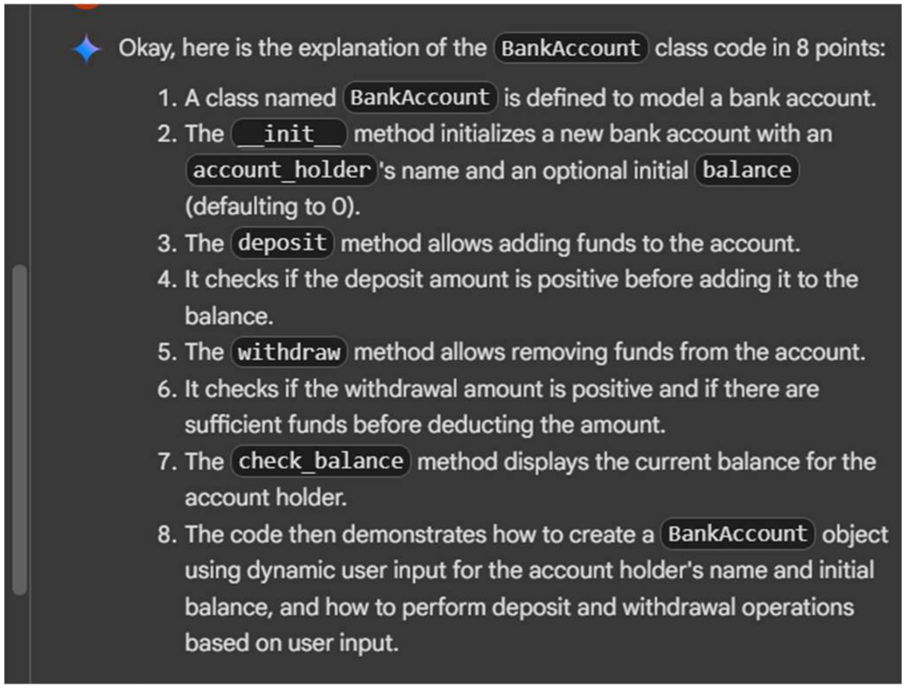
```python
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposit of ${amount} successful. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal of ${amount} successful. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f"Current balance for {self.account_holder}: ${self.balance}")

account_holder_name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))

user_account = BankAccount(account_holder_name, initial_balance)

user_account.check_balance()

deposit_amount = float(input("Enter amount to deposit: "))
user_account.deposit(deposit_amount)

withdraw_amount = float(input("Enter amount to withdraw: "))
user_account.withdraw(withdraw_amount)

user_account.check_balance()
```

Output:



```
Enter account holder name: pooja
Enter initial balance: 1200
Current balance for pooja: $1200.0
Enter amount to deposit: 120
Deposit of $120.0 successful. New balance: $1320.0
Enter amount to withdraw: 1400
Insufficient balance.
Current balance for pooja: $1320.0
```

Explanation:

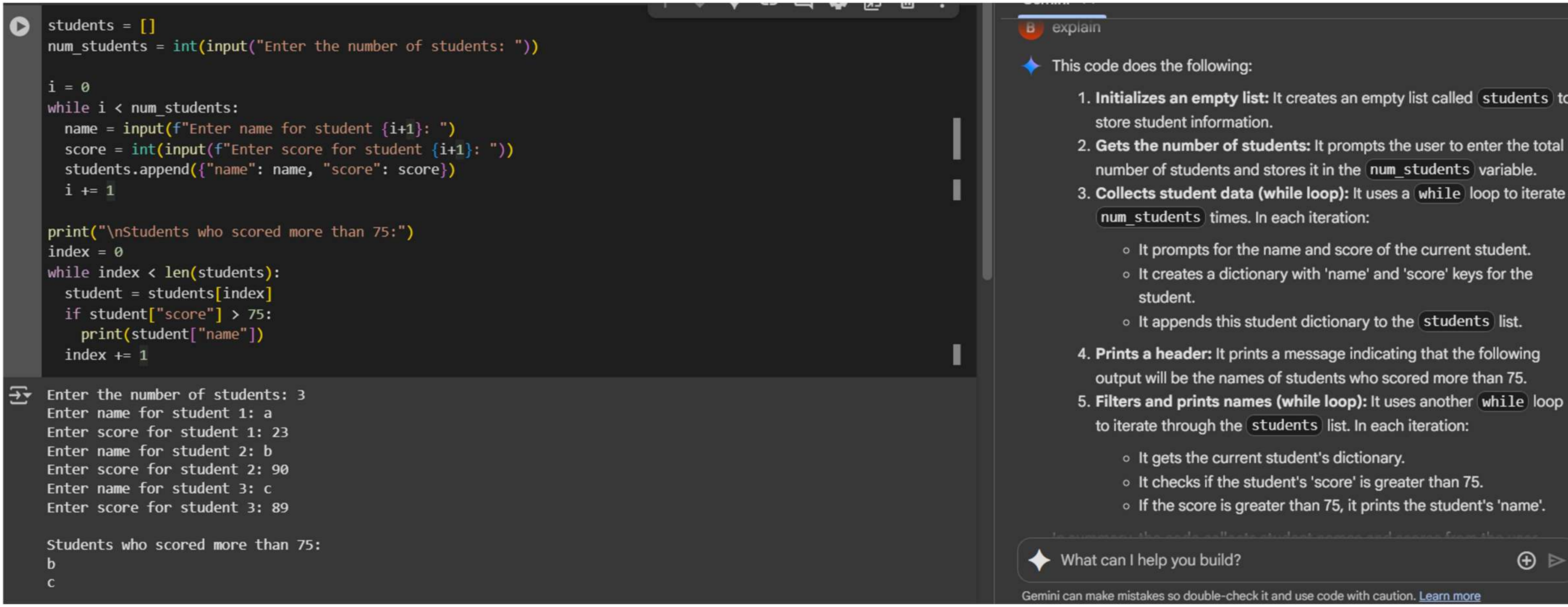


Okay, here is the explanation of the `BankAccount` class code in 8 points:

1. A class named `BankAccount` is defined to model a bank account.
2. The `__init__` method initializes a new bank account with an `account_holder`'s name and an optional initial `balance` (defaulting to 0).
3. The `deposit` method allows adding funds to the account.
4. It checks if the deposit amount is positive before adding it to the balance.
5. The `withdraw` method allows removing funds from the account.
6. It checks if the withdrawal amount is positive and if there are sufficient funds before deducting the amount.
7. The `check_balance` method displays the current balance for the account holder.
8. The code then demonstrates how to create a `BankAccount` object using dynamic user input for the account holder's name and initial balance, and how to perform deposit and withdrawal operations based on user input.

Task-4:

Prompt:write a python code to define a list of student dictionaries with key name and score and write a while loop to print the names of students who secured more than 75 by dynamic inputs by user



```
students = []
num_students = int(input("Enter the number of students: "))

i = 0
while i < num_students:
  name = input(f"Enter name for student {i+1}: ")
  score = int(input(f"Enter score for student {i+1}: "))
  students.append({"name": name, "score": score})
  i += 1

print("\nStudents who scored more than 75:")
index = 0
while index < len(students):
  student = students[index]
  if student["score"] > 75:
    print(student["name"])
  index += 1
```

```
Enter the number of students: 3
Enter name for student 1: a
Enter score for student 1: 23
Enter name for student 2: b
Enter score for student 2: 90
Enter name for student 3: c
Enter score for student 3: 89

Students who scored more than 75:
b
c
```

explain

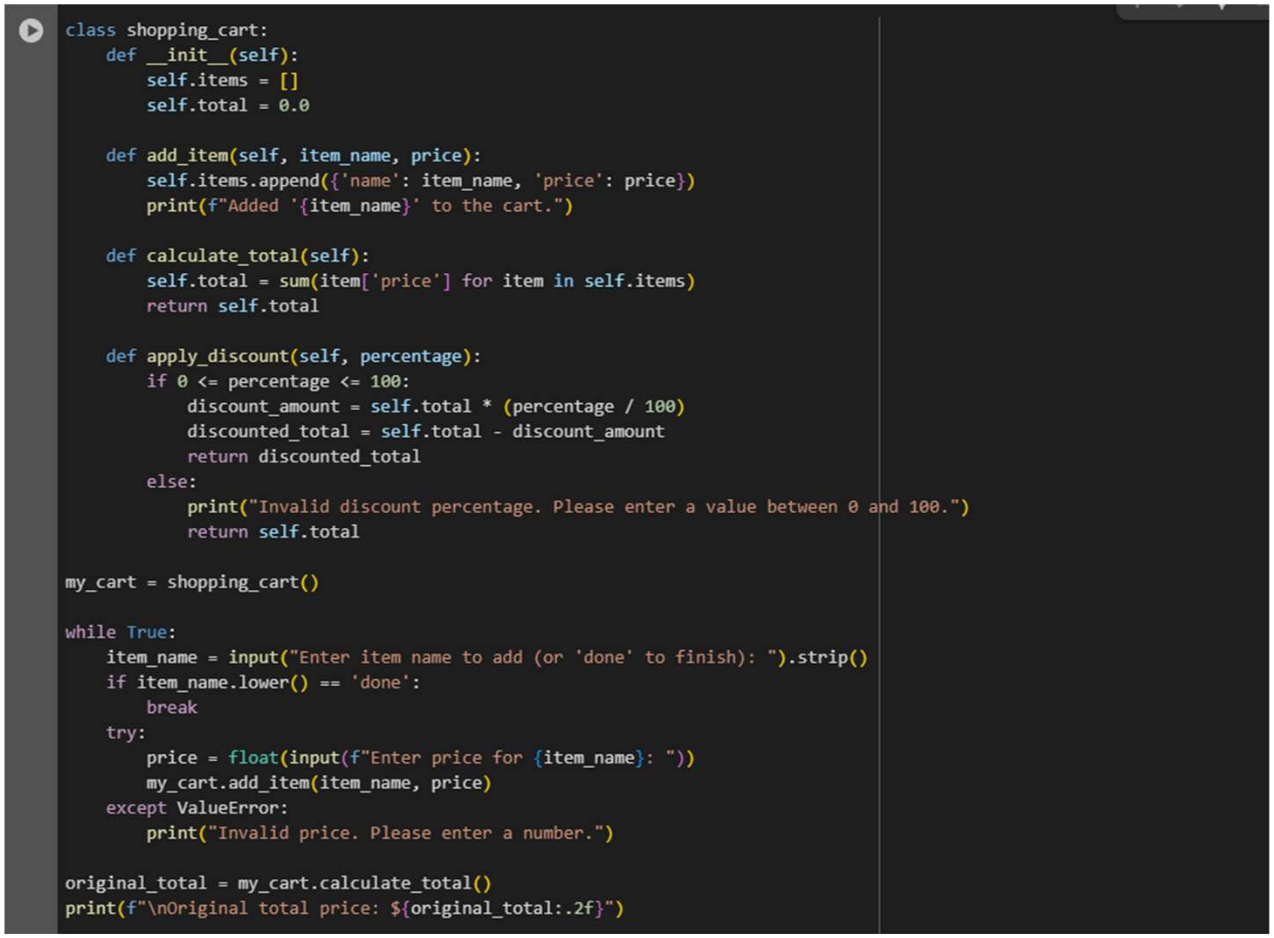
This code does the following:

1. **Initializes an empty list:** It creates an empty list called `students`
   store student information.
2. **Gets the number of students:** It prompts the user to enter the total number of students and stores it in the `num_students` variable.
3. **Collects student data (while loop):** It uses a `while` loop to iterate `num_students` times. In each iteration:
   - It prompts for the name and score of the current student.
   - It creates a dictionary with 'name' and 'score' keys for the student.
   - It appends this student dictionary to the `students` list.
4. **Prints a header:** It prints a message indicating that the following output will be the names of students who scored more than 75.
5. **Filters and prints names (while loop):** It uses another `while` loop to iterate through the `students` list. In each iteration:
   - It gets the current student's dictionary.
   - It checks if the student's 'score' is greater than 75.
   - If the score is greater than 75, it prints the student's 'name'.

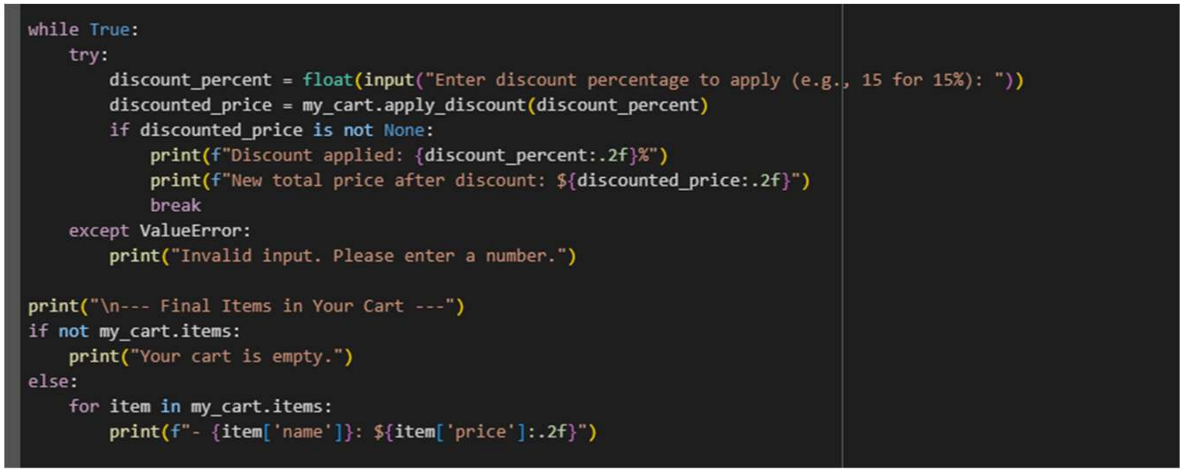
Gemini can make mistakes so double-check it and use code with caution. Learn more

Task-5:

Prompt: write a python code to begin writing a class shoppingcart with an empty items list and generate methods to add items ,remove items and use loop to calculate the total bill using discount condition by dynamic inputs by user

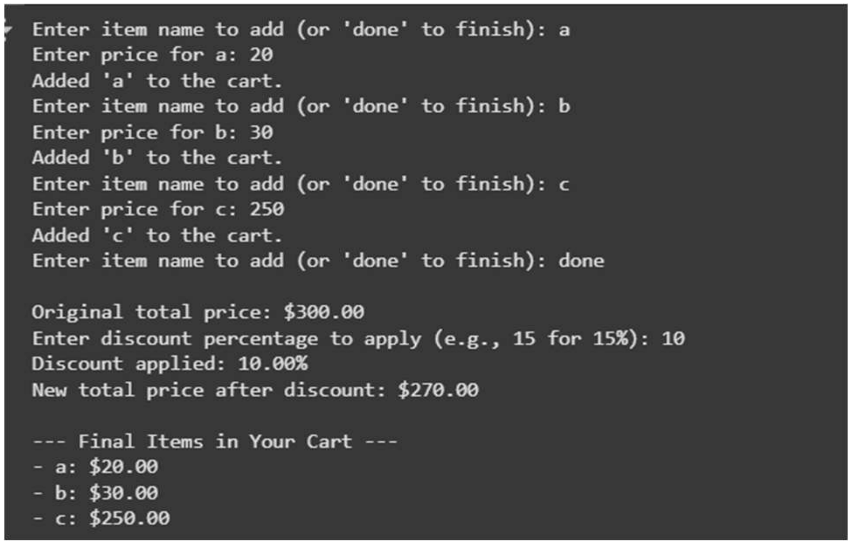


```
class shopping_cart:
    def __init__(self):
        self.items = []
        self.total = 0.0

    def add_item(self, item_name, price):
        self.items.append({'name': item_name, 'price': price})
        print(f"Added '{item_name}' to the cart.")

    def calculate_total(self):
        self.total = sum(item['price'] for item in self.items)
        return self.total

    def apply_discount(self, percentage):
        if 0 <= percentage <= 100:
            discount_amount = self.total * (percentage / 100)
            discounted_total = self.total - discount_amount
            return discounted_total
        else:
            print("Invalid discount percentage. Please enter a value between 0 and 100.")
            return self.total

my_cart = shopping_cart()

while True:
    item_name = input("Enter item name to add (or 'done' to finish): ").strip()
    if item_name.lower() == 'done':
        break
    try:
        price = float(input(f"Enter price for {item_name}: "))
        my_cart.add_item(item_name, price)
    except ValueError:
        print("Invalid price. Please enter a number.")

original_total = my_cart.calculate_total()
print(f"\nOriginal total price: ${original_total:.2f}")
```



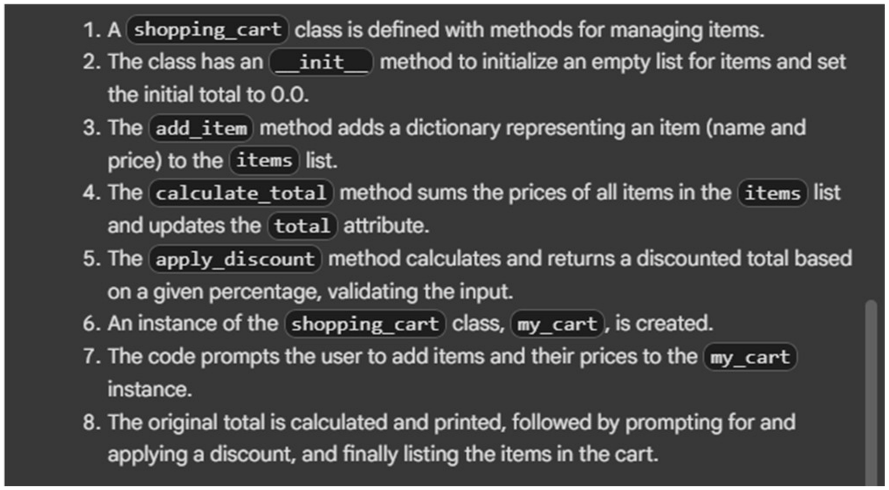
```python
while True:
    try:
        discount_percent = float(input("Enter discount percentage to apply (e.g., 15 for 15%): "))
        discounted_price = my_cart.apply_discount(discount_percent)
        if discounted_price is not None:
            print(f"Discount applied: {discount_percent:.2f}%")
            print(f"New total price after discount: ${discounted_price:.2f}")
            break
    except ValueError:
        print("Invalid input. Please enter a number.")

print("\n--- Final Items in Your Cart ---")
if not my_cart.items:
    print("Your cart is empty.")
else:
    for item in my_cart.items:
        print(f"- {item['name']}: ${item['price']:.2f}")
```

Output:



```
Enter item name to add (or 'done' to finish): a
Enter price for a: 20
Added 'a' to the cart.
Enter item name to add (or 'done' to finish): b
Enter price for b: 30
Added 'b' to the cart.
Enter item name to add (or 'done' to finish): c
Enter price for c: 250
Added 'c' to the cart.
Enter item name to add (or 'done' to finish): done

Original total price: $300.00
Enter discount percentage to apply (e.g., 15 for 15%): 10
Discount applied: 10.00%
New total price after discount: $270.00

--- Final Items in Your Cart ---
- a: $20.00
- b: $30.00
- c: $250.00
```

Explanation:



1. A `shopping_cart` class is defined with methods for managing items.
2. The class has an `__init__` method to initialize an empty list for items and set the initial total to 0.0.
3. The `add_item` method adds a dictionary representing an item (name and price) to the `items` list.
4. The `calculate_total` method sums the prices of all items in the `items` list and updates the `total` attribute.
5. The `apply_discount` method calculates and returns a discounted total based on a given percentage, validating the input.
6. An instance of the `shopping_cart` class, `my_cart`, is created.
7. The code prompts the user to add items and their prices to the `my_cart` instance.
8. The original total is calculated and printed, followed by prompting for and applying a discount, and finally listing the items in the cart.