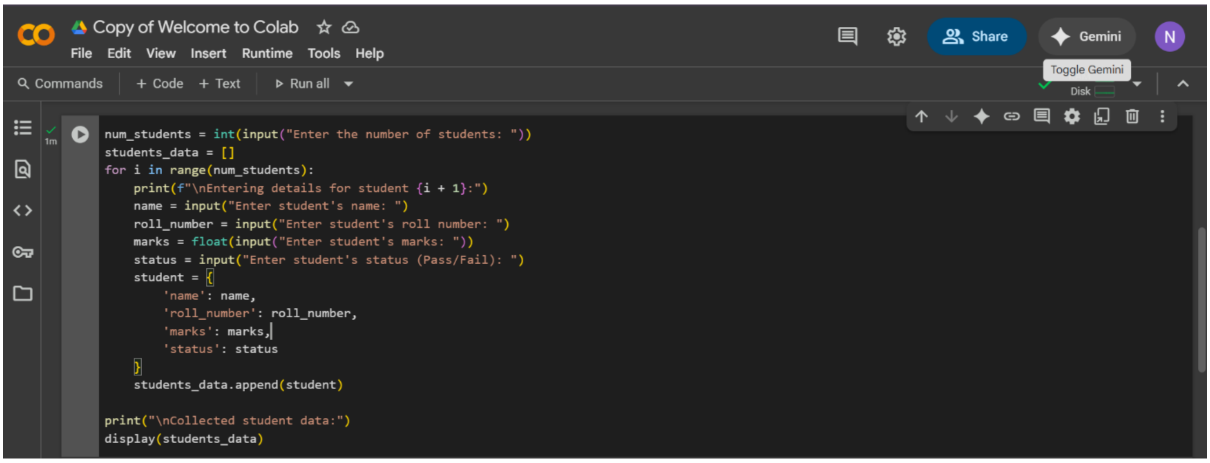
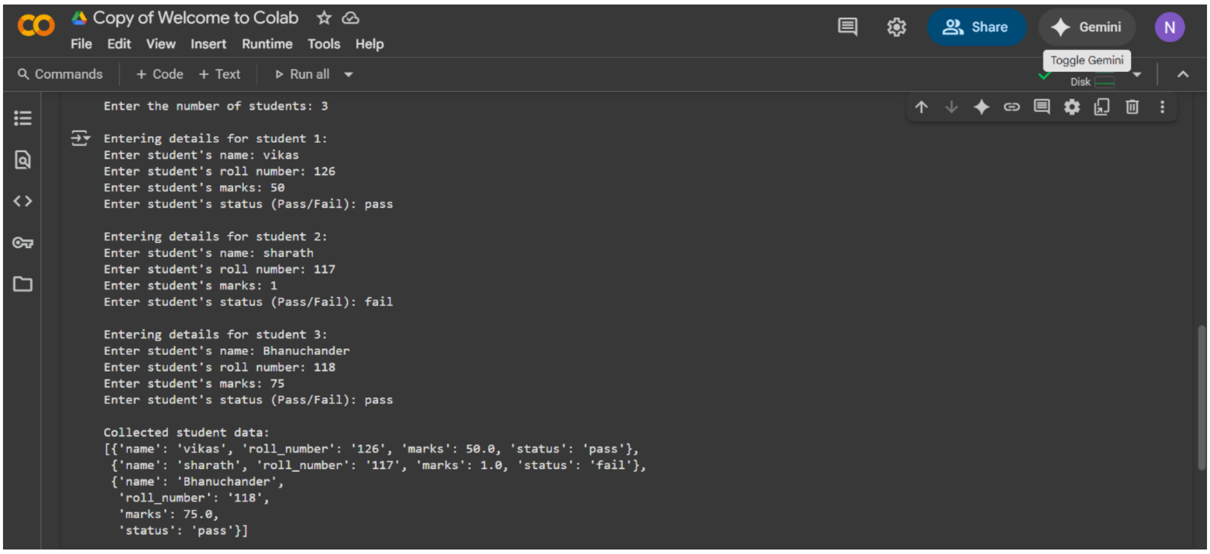
Assignment-6.4

Task-1; • Start a Python class named Student with attributes name, roll_number, and marks. Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average

Code and output;

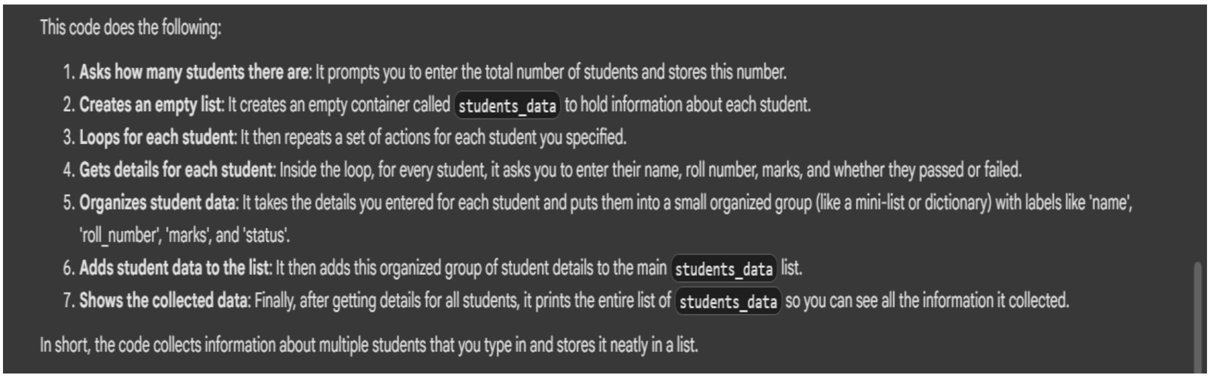


```
num_students = int(input("Enter the number of students: "))
students_data = []
for i in range(num_students):
    print(f"\nEntering details for student {i + 1}:")
    name = input("Enter student's name: ")
    roll_number = input("Enter student's roll number: ")
    marks = float(input("Enter student's marks: "))
    status = input("Enter student's status (Pass/Fail): ")
    student = {
        'name': name,
        'roll_number': roll_number,
        'marks': marks,
        'status': status
    }
    students_data.append(student)

print("\nCollected student data:")
display(students_data)
```



```
Enter the number of students: 3

Entering details for student 1:
Enter student's name: vikas
Enter student's roll number: 126
Enter student's marks: 50
Enter student's status (Pass/Fail): pass

Entering details for student 2:
Enter student's name: sharath
Enter student's roll number: 117
Enter student's marks: 1
Enter student's status (Pass/Fail): fail

Entering details for student 3:
Enter student's name: Bhanuchander
Enter student's roll number: 118
Enter student's marks: 75
Enter student's status (Pass/Fail): pass

Collected student data:
[{'name': 'vikas', 'roll_number': '126', 'marks': 50.0, 'status': 'pass'},
 {'name': 'sharath', 'roll_number': '117', 'marks': 1.0, 'status': 'fail'},
 {'name': 'Bhanuchander',
  'roll_number': '118',
  'marks': 75.0,
  'status': 'pass'}]
```

Explanation;



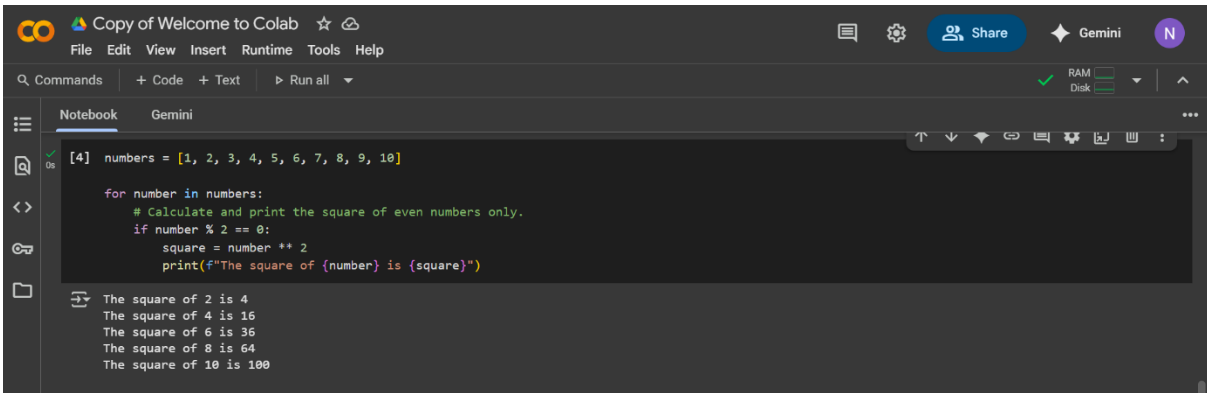
This code does the following:

1. **Asks how many students there are:** It prompts you to enter the total number of students and stores this number.
2. **Creates an empty list:** It creates an empty container called `students_data` to hold information about each student.
3. **Loops for each student:** It then repeats a set of actions for each student you specified.
4. **Gets details for each student:** Inside the loop, for every student, it asks you to enter their name, roll number, marks, and whether they passed or failed.
5. **Organizes student data:** It takes the details you entered for each student and puts them into a small organized group (like a mini-list or dictionary) with labels like 'name', 'roll_number', 'marks', and 'status'.
6. **Adds student data to the list:** It then adds this organized group of student details to the main `students_data` list.
7. **Shows the collected data:** Finally, after getting details for all students, it prints the entire list of `students_data` so you can see all the information it collected.

In short, the code collects information about multiple students that you type in and stores it neatly in a list.

Task-2; • Write the first two lines of a for loop to iterate through a list of numbers. Use a comment prompt to let Copilot suggest how to calculate and print the square of even numbers only.

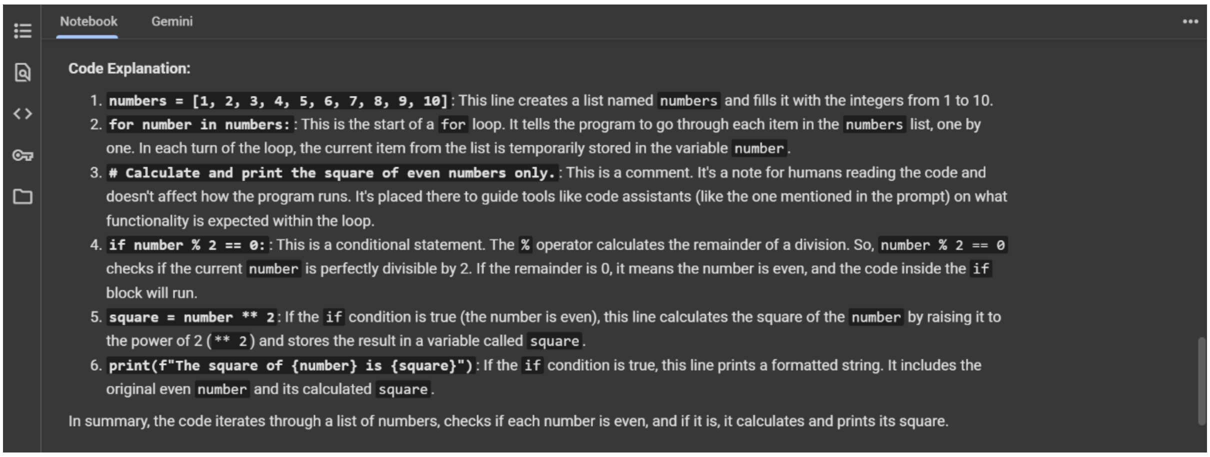
Code and output;



```
[4]  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

     for number in numbers:
         # Calculate and print the square of even numbers only.
         if number % 2 == 0:
             square = number ** 2
             print(f"The square of {number} is {square}")
```

```
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
```

Explanation;



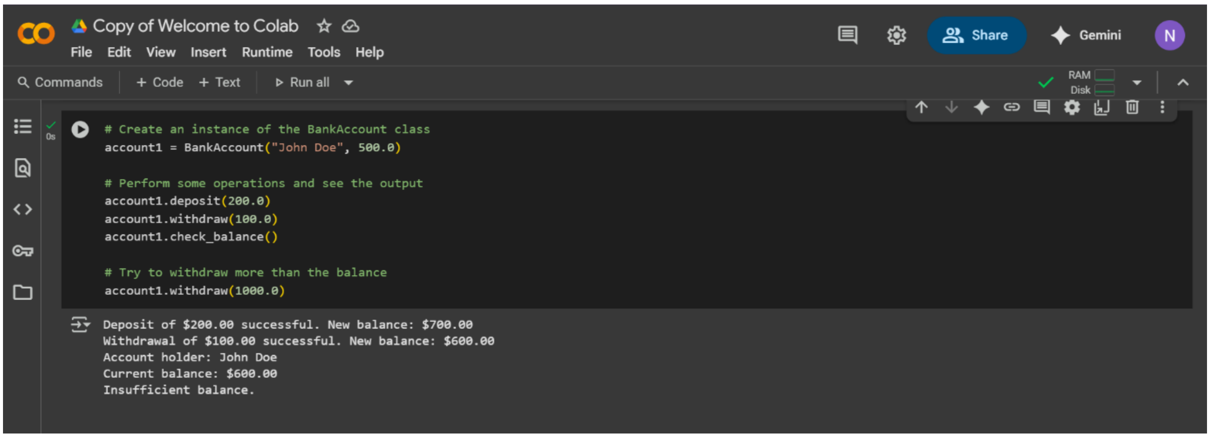
Code Explanation:

1. **`numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`**: This line creates a list named `numbers` and fills it with the integers from 1 to 10.
2. **`for number in numbers:`**: This is the start of a `for` loop. It tells the program to go through each item in the `numbers` list, one by one. In each turn of the loop, the current item from the list is temporarily stored in the variable `number`.
3. **`# Calculate and print the square of even numbers only.`**: This is a comment. It's a note for humans reading the code and doesn't affect how the program runs. It's placed there to guide tools like code assistants (like the one mentioned in the prompt) on what functionality is expected within the loop.
4. **`if number % 2 == 0:`**: This is a conditional statement. The `%` operator calculates the remainder of a division. So, `number % 2 == 0` checks if the current `number` is perfectly divisible by 2. If the remainder is 0, it means the number is even, and the code inside the `if` block will run.
5. **`square = number ** 2`**: If the `if` condition is true (the number is even), this line calculates the square of the `number` by raising it to the power of 2 (`** 2`) and stores the result in a variable called `square`.
6. **`print(f"The square of {number} is {square}")`**: If the `if` condition is true, this line prints a formatted string. It includes the original even `number` and its calculated `square`.

In summary, the code iterates through a list of numbers, checks if each number is even, and if it is, it calculates and prints its square.

Task-3; Create a class called BankAccount with attributes account_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance.

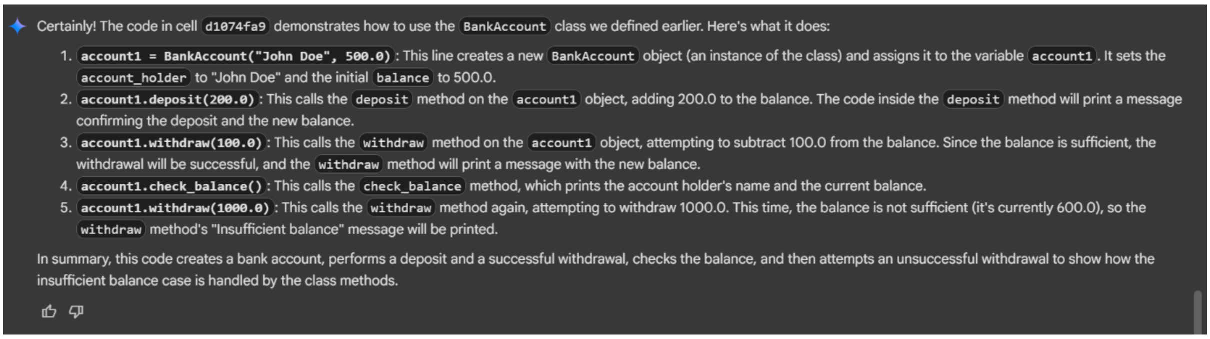
Code and output;



```
# Create an instance of the BankAccount class
account1 = BankAccount("John Doe", 500.0)

# Perform some operations and see the output
account1.deposit(200.0)
account1.withdraw(100.0)
account1.check_balance()

# Try to withdraw more than the balance
account1.withdraw(1000.0)
```

```
Deposit of $200.00 successful. New balance: $700.00
Withdrawal of $100.00 successful. New balance: $600.00
Account holder: John Doe
Current balance: $600.00
Insufficient balance.
```

Explanation;



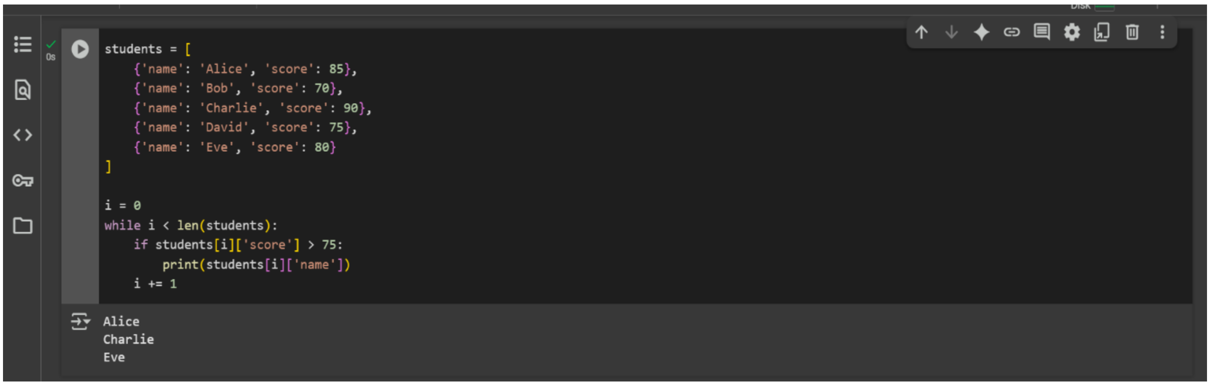
Certainly! The code in cell `d1074fa9` demonstrates how to use the `BankAccount` class we defined earlier. Here's what it does:

1. **`account1 = BankAccount("John Doe", 500.0)`**: This line creates a new `BankAccount` object (an instance of the class) and assigns it to the variable `account1`. It sets the `account_holder` to "John Doe" and the initial `balance` to 500.0.
2. **`account1.deposit(200.0)`**: This calls the `deposit` method on the `account1` object, adding 200.0 to the balance. The code inside the `deposit` method will print a message confirming the deposit and the new balance.
3. **`account1.withdraw(100.0)`**: This calls the `withdraw` method on the `account1` object, attempting to subtract 100.0 from the balance. Since the balance is sufficient, the withdrawal will be successful, and the `withdraw` method will print a message with the new balance.
4. **`account1.check_balance()`**: This calls the `check_balance` method, which prints the account holder's name and the current balance.
5. **`account1.withdraw(1000.0)`**: This calls the `withdraw` method again, attempting to withdraw 1000.0. This time, the balance is not sufficient (it's currently 600.0), so the `withdraw` method's "Insufficient balance" message will be printed.

In summary, this code creates a bank account, performs a deposit and a successful withdrawal, checks the balance, and then attempts an unsuccessful withdrawal to show how the insufficient balance case is handled by the class methods.

Task-4; Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

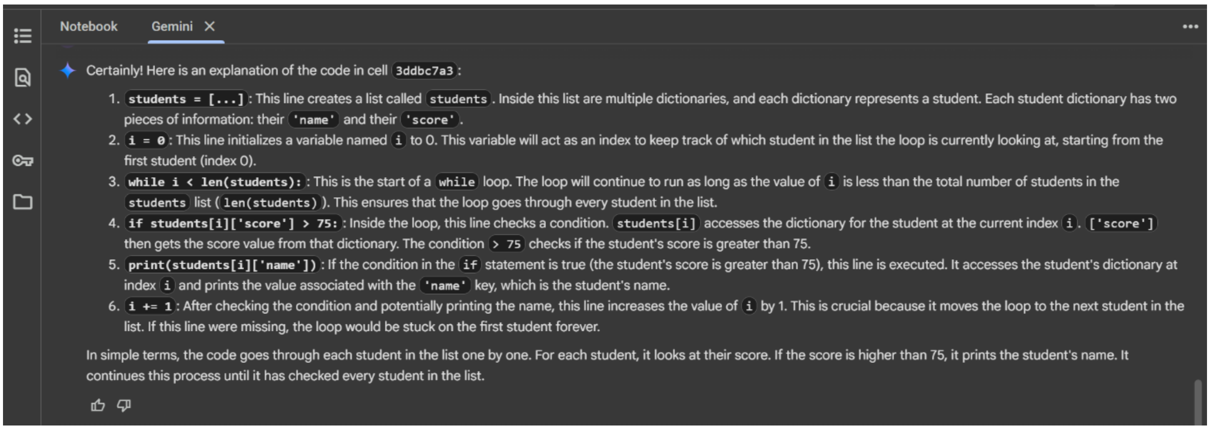
Code and output;



```
students = [
    {'name': 'Alice', 'score': 85},
    {'name': 'Bob', 'score': 70},
    {'name': 'Charlie', 'score': 90},
    {'name': 'David', 'score': 75},
    {'name': 'Eve', 'score': 80}
]

i = 0
while i < len(students):
    if students[i]['score'] > 75:
        print(students[i]['name'])
    i += 1
```

```
Alice
Charlie
Eve
```

Explanation;



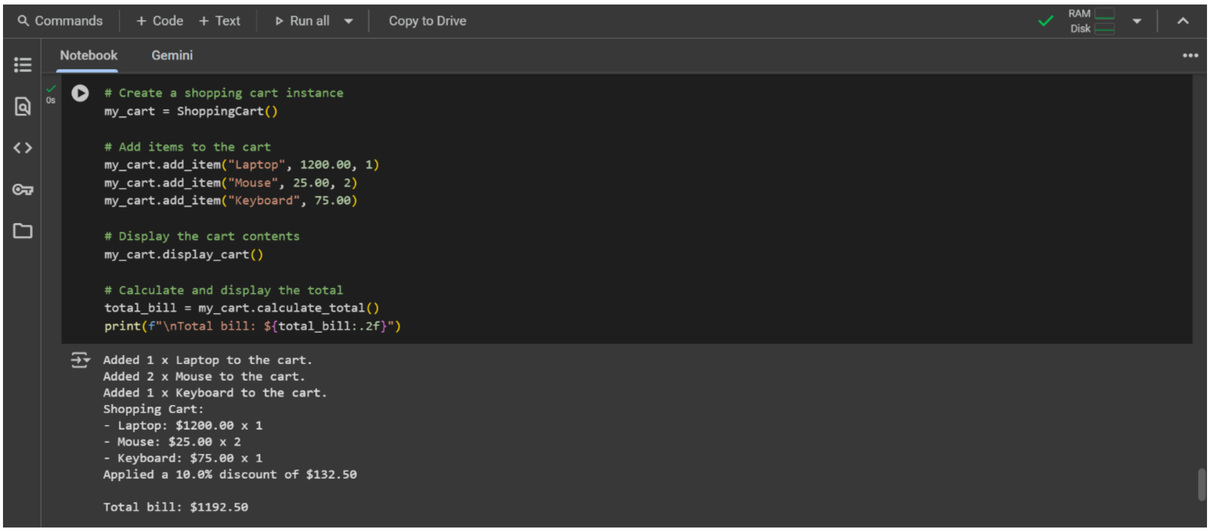
Certainly! Here is an explanation of the code in cell `3ddbc7a3`:

1. `students = [...]`: This line creates a list called `students`. Inside this list are multiple dictionaries, and each dictionary represents a student. Each student dictionary has two pieces of information: their `'name'` and their `'score'`.
2. `i = 0`: This line initializes a variable named `i` to 0. This variable will act as an index to keep track of which student in the list the loop is currently looking at, starting from the first student (index 0).
3. `while i < len(students):`: This is the start of a `while` loop. The loop will continue to run as long as the value of `i` is less than the total number of students in the `students` list (`len(students)`). This ensures that the loop goes through every student in the list.
4. `if students[i]['score'] > 75:`: Inside the loop, this line checks a condition. `students[i]` accesses the dictionary for the student at the current index `i`. `['score']` then gets the score value from that dictionary. The condition `> 75` checks if the student's score is greater than 75.
5. `print(students[i]['name'])`: If the condition in the `if` statement is true (the student's score is greater than 75), this line is executed. It accesses the student's dictionary at index `i` and prints the value associated with the `'name'` key, which is the student's name.
6. `i += 1`: After checking the condition and potentially printing the name, this line increases the value of `i` by 1. This is crucial because it moves the loop to the next student in the list. If this line were missing, the loop would be stuck on the first student forever.

In simple terms, the code goes through each student in the list one by one. For each student, it looks at their score. If the score is higher than 75, it prints the student's name. It continues this process until it has checked every student in the list.

Task-5; • Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.

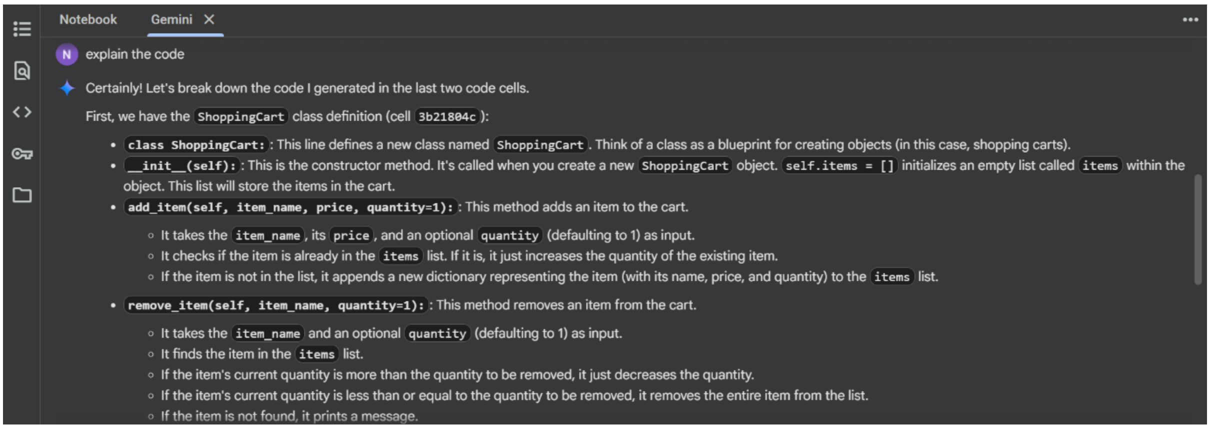
Code and output;



```python
# Create a shopping cart instance
my_cart = ShoppingCart()

# Add items to the cart
my_cart.add_item("Laptop", 1200.00, 1)
my_cart.add_item("Mouse", 25.00, 2)
my_cart.add_item("Keyboard", 75.00)

# Display the cart contents
my_cart.display_cart()

# Calculate and display the total
total_bill = my_cart.calculate_total()
print(f"\nTotal bill: ${total_bill:.2f}")
```

```
Added 1 x Laptop to the cart.
Added 2 x Mouse to the cart.
Added 1 x Keyboard to the cart.
Shopping Cart:
- Laptop: $1200.00 x 1
- Mouse: $25.00 x 2
- Keyboard: $75.00 x 1
Applied a 10.0% discount of $132.50

Total bill: $1192.50
```

Explanation;

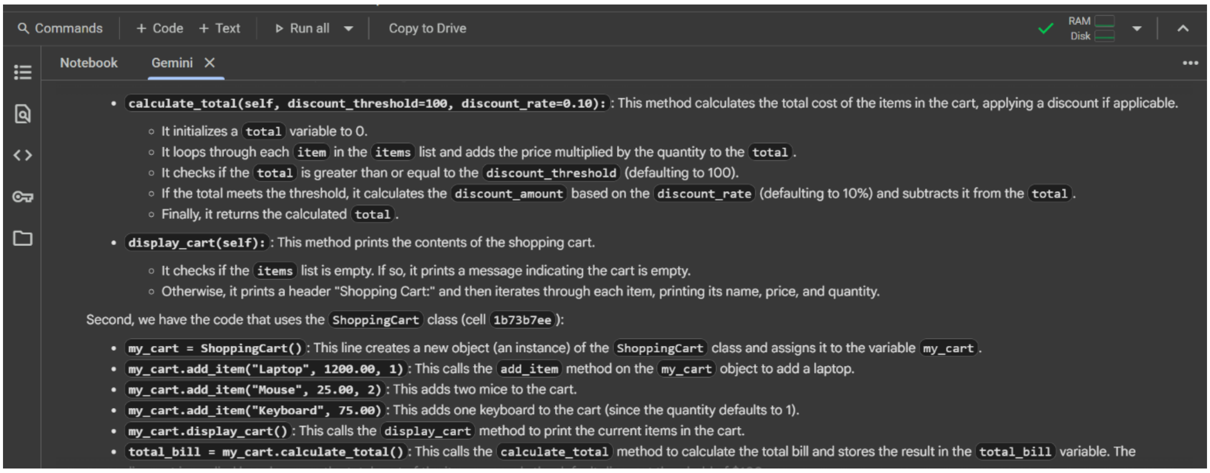


N explain the code

Certainly! Let's break down the code I generated in the last two code cells.

First, we have the `ShoppingCart` class definition (cell `3b21804c`):

- `class ShoppingCart:` : This line defines a new class named `ShoppingCart`. Think of a class as a blueprint for creating objects (in this case, shopping carts).
- `__init__(self):` : This is the constructor method. It's called when you create a new `ShoppingCart` object. `self.items = []` initializes an empty list called `items` within the object. This list will store the items in the cart.
- `add_item(self, item_name, price, quantity=1):` : This method adds an item to the cart.
  - It takes the `item_name`, its `price`, and an optional `quantity` (defaulting to 1) as input.
  - It checks if the item is already in the `items` list. If it is, it just increases the quantity of the existing item.
  - If the item is not in the list, it appends a new dictionary representing the item (with its name, price, and quantity) to the `items` list.
- `remove_item(self, item_name, quantity=1):` : This method removes an item from the cart.
  - It takes the `item_name` and an optional `quantity` (defaulting to 1) as input.
  - It finds the item in the `items` list.
  - If the item's current quantity is more than the quantity to be removed, it just decreases the quantity.
  - If the item's current quantity is less than or equal to the quantity to be removed, it removes the entire item from the list.
  - If the item is not found, it prints a message.



- `calculate_total(self, discount_threshold=100, discount_rate=0.10):` : This method calculates the total cost of the items in the cart, applying a discount if applicable.
  - It initializes a `total` variable to 0.
  - It loops through each `item` in the `items` list and adds the price multiplied by the quantity to the `total`.
  - It checks if the `total` is greater than or equal to the `discount_threshold` (defaulting to 100).
  - If the total meets the threshold, it calculates the `discount_amount` based on the `discount_rate` (defaulting to 10%) and subtracts it from the `total`.
  - Finally, it returns the calculated `total`.
- `display_cart(self):` : This method prints the contents of the shopping cart.
  - It checks if the `items` list is empty. If so, it prints a message indicating the cart is empty.
  - Otherwise, it prints a header "Shopping Cart:" and then iterates through each item, printing its name, price, and quantity.

Second, we have the code that uses the `ShoppingCart` class (cell `1b73b7ee`):

- `my_cart = ShoppingCart()` : This line creates a new object (an instance) of the `ShoppingCart` class and assigns it to the variable `my_cart`.
- `my_cart.add_item("Laptop", 1200.00, 1)` : This calls the `add_item` method on the `my_cart` object to add a laptop.
- `my_cart.add_item("Mouse", 25.00, 2)` : This adds two mice to the cart.
- `my_cart.add_item("Keyboard", 75.00)` : This adds one keyboard to the cart (since the quantity defaults to 1).
- `my_cart.display_cart()` : This calls the `display_cart` method to print the current items in the cart.
- `total_bill = my_cart.calculate_total()` : This calls the `calculate_total` method to calculate the total bill and stores the result in the `total_bill` variable. The