

.ASSIGNMENT-6.4.

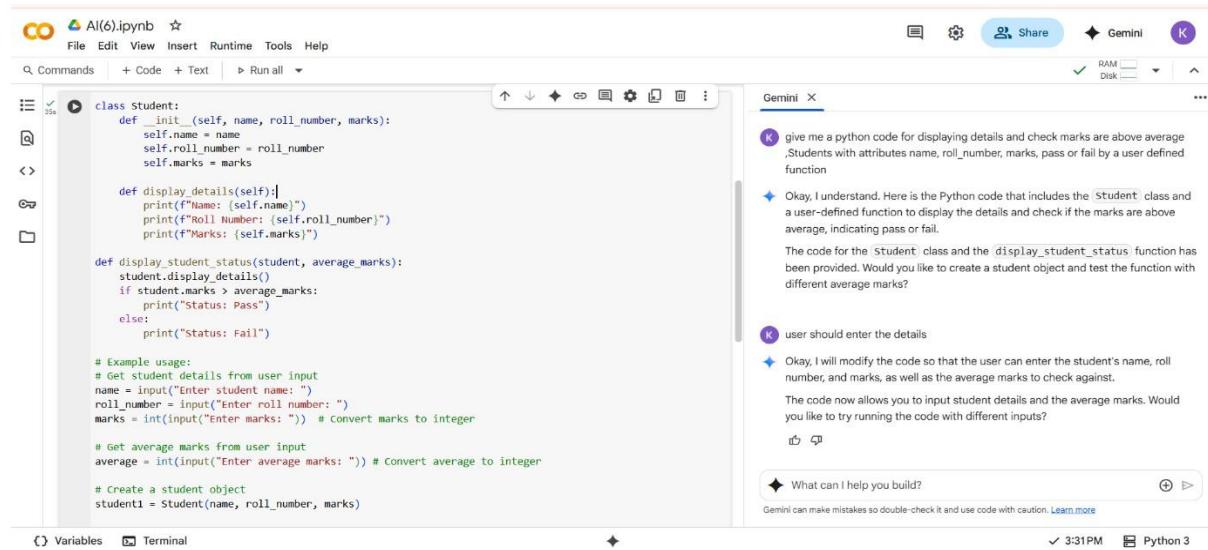
NAME: K.SRI SAI HARSHA KISHORE
2403A52133

Task 1:

>Give me a python code for displaying details and check marks are above average ,Students with attributes name, roll-number, marks, pass or fail.

>User should enter the details.

Code and Output:



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for a `Student` class and a `display_student_status` function. The code includes input prompts for name, roll number, and marks, and outputs the student's details and status (Pass/Fail). A Gemini AI chat window is open, responding to the user's request for a code that displays details and checks if marks are above average. It provides the requested code and suggests modifying it to allow user input for name, roll number, and marks.

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

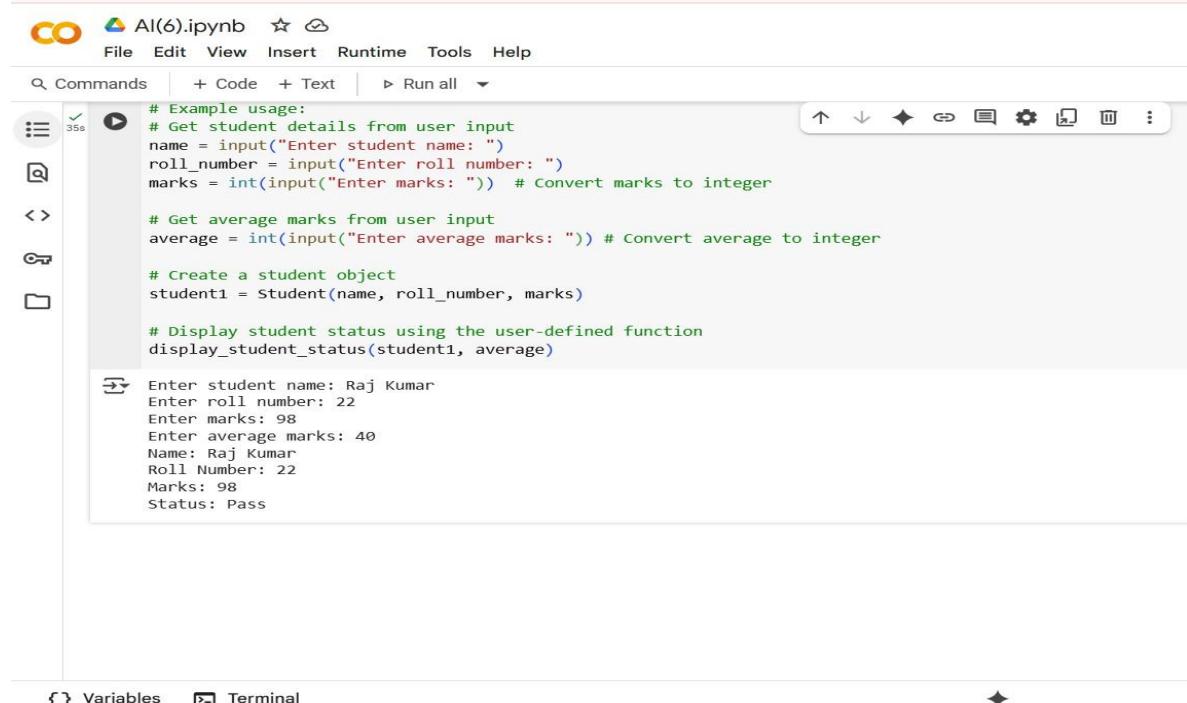
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

def display_student_status(student, average_marks):
    student.display_details()
    if student.marks > average_marks:
        print("Status: Pass")
    else:
        print("Status: Fail")

# Example usage:
# Get student details from user input
name = input("Enter student name: ")
roll_number = input("Enter roll number: ")
marks = int(input("Enter marks: ")) # Convert marks to integer

# Get average marks from user input
average = int(input("Enter average marks: ")) # Convert average to integer

# Create a student object
student1 = Student(name, roll_number, marks)
```



The screenshot shows the execution of the Python code from the previous screenshot. The code prompts for student details and average marks, then prints the student's details and status. The output shows a sample run where the user enters 'Raj Kumar' as the name, '22' as the roll number, '98' as the marks, and '40' as the average marks. The code correctly identifies the status as 'Pass'.

```
# Example usage:
# Get student details from user input
name = input("Enter student name: ")
roll_number = input("Enter roll number: ")
marks = int(input("Enter marks: ")) # Convert marks to integer

# Get average marks from user input
average = int(input("Enter average marks: ")) # Convert average to integer

# Create a student object
student1 = Student(name, roll_number, marks)

# Display student status using the user-defined function
display_student_status(student1, average)
```

Enter student name: Raj Kumar
Enter roll number: 22
Enter marks: 98
Enter average marks: 40
Name: Raj Kumar
Roll Number: 22
Marks: 98
Status: Pass

Explanation:

Gemini X

K explanation

◆ Certainly, I can explain the content of the cells you have visible.

Cell 1 (id: d38aac2b): This is a Markdown cell that provides a summary of the "ShoppingCart" task, key findings, and insights or next steps.

- It summarizes that the `ShoppingCart` class was successfully created with methods for adding, removing, and calculating the total with discounts.
- It lists key findings, such as the successful implementation of each method and the details of the conditional discounts applied (10% for quantity > 5 and 5% for price > \$50).
- It suggests insights or next steps, including adding validation to the `add_item` method and considering more complex discount rules.

Cell 2 (id: dbe3c467): This cell defines a Python class named `Student`.

- The `__init__` method is the constructor of the class. It initializes a student object with their `name`, `roll_number`, and `marks`. It also takes `average_marks` as input during initialization and calculates whether the student passed by calling the `is_above_average` method. The result of this check is stored in the `self.passed` attribute.
- The `display_details` method prints the student's name, roll number, marks, and whether they passed (Yes or No) based on the value of the `self.passed` attribute.
- The `is_above_average` method takes `average_marks` as input and returns `True` if the student's `marks` are greater than `average_marks`, and `False` otherwise.
- The commented-out lines at the end of the cell show example usage of how to create a `Student` object and call the `display_details` method.

✍️ ↻

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Task 2:

> Write a python program to creating the list of two numbers.

> Write the first two lines of a for loop to iterate through the list.

> Calculates and print the square of even numbers only

Code and Output:

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Gemini X

K write a python program to creating the list of two numbers

◆ Certainly, here is a simple Python code to create a list containing two numbers.

I have created a list with two numbers. Would you like to perform any operations on this list, like adding more numbers or accessing specific elements?

Gemini X

K write the first two lines of a for loop to iterate through the list

◆ Okay, here are the first two lines of a `for` loop to iterate through the list `my_list`:

It seems there was a `SyntaxError` because the `for` loop is incomplete. You need to add some code inside the loop to process each number. What would you like to do with each number in the list? For example, you could print each number.

✍️ ↻

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 3:39PM Python 3

Explanation:

Gemini X

K explanation

◆ Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 102e44c1): This cell defines a `Student` class and a function `display_student_status`.

- The `Student` class has an `__init__` method to initialize a student with a name, roll number, and marks.
- It also has a `display_details` method to print the student's name, roll number, and marks.
- The `display_student_status` function takes a `Student` object and an `average_marks` value as input. It first displays the student's details using the `display_details` method and then prints whether the student "Pass" or "Fail" based on whether their marks are greater than the provided `average_marks`.
- The rest of the code in this cell is an example of how to use the class and function, including getting input from the user for student details and average marks, creating a `Student` object, and then calling `display_student_status`.

Cell 2 (id: f2de40b8): This cell creates a Python list named `my_list` with four integer values: 10, 22, 3, and 20. Then, it prints the entire list to the console.

Cell 3 (id: 160679fa): This cell contains a `for` loop that iterates through each `number` in the `my_list`.

- Inside the loop, it checks if the current `number` is even using the modulo operator (%). If `number % 2 == 0` is true, it means the number is even.
- If the number is even, it calculates the square of the number (`number ** 2`) and stores it in the `square` variable.
- Finally, it prints the calculated `square` value to the console. This means only the squares of the even numbers from `my_list` will be printed.

✍️ ↻

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

3:39PM Python 3

Task 3:

>Write the skeleton with attributes to generate methods def deposit(self.amount):def withdraw(self.amount) and a method to check the balance.

>Test the class by creating objects.

Code and Output:

The screenshot shows a Jupyter Notebook interface with two panes. The left pane contains Python code for a class named `ClassName`. The right pane is a chat window with a GPT-4 AI named Gemini.

Code (Left Pane):

```
[21] class ClassName:
    def __init__(self, attribute1, attribute2, initial_balance=0):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
        self.balance = initial_balance # Added balance attribute

    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 funds.")
        else:
            print("Withdrawal amount must be positive.")

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

    # Add other methods here

# Create an object of the ClassName
# Replace 'value1' and 'value2' with actual values for attribute1 and attribute2
```

GPT-4 Chat (Right Pane):

- K write the skeleton with attributes
- OKay, here is a skeleton of a Python class with an `__init__` method to define and initialize attributes. You can replace `ClassName` and the attribute names with your desired names.
- I have provided a skeleton for a Python class with attributes. Would you like to add more methods to this class or create an object from this class?
- K to generate methods def deposit(self.amount):def withdraw(self.amount) and a method to check the balance
- OKay, I will add the `deposit`, `withdraw`, and `check_balance` methods to the class skeleton. I'll also include a `balance` attribute to keep track of the account balance.
- I have added the `deposit`, `withdraw`, and `check_balance` methods to the class. Would you like to create an instance of this class and test these methods?
- K test the class by creating objects
- Great! Let's test the `ClassName` by creating an object and using the methods.
- I have created an object of the `ClassName` and tested the `deposit`, `withdraw`, and
- What can I help you build?

The screenshot shows a Jupyter Notebook interface with two panes. The left pane contains Python code for a class named `ClassName`. The right pane is a chat window with a GPT-4 AI named Gemini.

Code (Left Pane):

```
[22] # Create an object of the ClassName
# Replace 'value1' and 'value2' with actual values for attribute1 and attribute2
my_object = ClassName('value1', 'value2', initial_balance=100)

# Test the check_balance method
my_object.check_balance()

# Test the deposit method
my_object.deposit(50)

# Test the withdraw method
my_object.withdraw(30)

# Test withdrawal with insufficient funds
my_object.withdraw(200)

# Check the balance again
my_object.check_balance()
```

GPT-4 Chat (Right Pane):

- K write the skeleton with attributes
- OKay, here is a skeleton of a Python class with an `__init__` method to define and initialize attributes. You can replace `ClassName` and the attribute names with your desired names.
- I have provided a skeleton for a Python class with attributes. Would you like to add more methods to this class or create an object from this class?
- K to generate methods def deposit(self.amount):def withdraw(self.amount) and a method to check the balance
- OKay, I will add the `deposit`, `withdraw`, and `check_balance` methods to the class skeleton. I'll also include a `balance` attribute to keep track of the account balance.
- I have added the `deposit`, `withdraw`, and `check_balance` methods to the class. Would you like to create an instance of this class and test these methods?
- K test the class by creating objects
- Great! Let's test the `ClassName` by creating an object and using the methods.
- I have created an object of the `ClassName` and tested the `deposit`, `withdraw`, and
- What can I help you build?

Explanation:

Gemini X

K explanation

◆ Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 160679fa): This cell contains a `for` loop that iterates through each `number` in the list `my_list` (which was defined in a previous cell).

- Inside the loop, it checks if the current `number` is even using the modulo operator (`%`). If `number % 2 == 0` is true, it means the number is even.
- If the number is even, it calculates the square of the number (`number ** 2`) and stores it in the `square` variable.
- Finally, it prints the calculated `square` value to the console. This means only the squares of the even numbers from `my_list` will be printed.

Cell 2 (id: ecf83fa1): This cell defines a Python class named `ClassName`.

- The `__init__` method is the constructor of the class. It initializes the object with three attributes: `attribute1`, `attribute2`, and `balance` (which has a default value of 0 if not provided).
- The `deposit` method takes an `amount` as input and adds it to the `balance` if the amount is positive. It then prints a message indicating the successful deposit and the new balance.
- The `withdraw` method takes an `amount` as input. If the amount is positive and there are sufficient funds (`self.balance >= amount`), it subtracts the amount from the `balance` and prints a success message with the new balance. Otherwise, it prints an "Insufficient funds" message.
- The `check_balance` method simply prints the current value of the `balance` attribute.

Cell 3 (id: 6ea779a8): This cell demonstrates how to use the `ClassName` defined in the previous cell.

- It creates an object named `my_object` from the `ClassName`, providing values for `attribute1`, `attribute2`, and an `initial_balance` of 100.
- It then calls the `check_balance()` method to display the initial balance.
- It calls the `deposit(50)` method to add 50 to the balance and shows the updated balance.
- It calls the `withdraw(30)` method to subtract 30 from the balance and shows the updated balance.
- It attempts to call `withdraw(200)` to demonstrate the "Insufficient funds" message because the current balance is less than 200.
- Finally, it calls `check_balance()` again to show the final balance after the operations.

◆ What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Task 4:

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

Code and Output:

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Define a list of student dictionaries
students = [
 {"name": "Alice", "score": 85},
 {"name": "Bob", "score": 70},
 {"name": "Charlie", "score": 92},
 {"name": "David", "score": 65},
 {"name": "Eve", "score": 78},
]

Initialize an index for the while loop
i = 0

Use a while loop to iterate through the list
while i < len(students):
 student = students[i]
 if student["score"] > 75:
 print(student["name"])
 i += 1

Alice
Charlie
Eve

Gemini X

K Define a list of student dictionaries with keys name and score and to write a while loop to print the names of students who scored more than 75.

◆ Okay, I can help with that. Here is the Python code to define a list of student dictionaries and use a `while` loop to print the names of students who scored more than 75.

I have created the list of student dictionaries and added the `while` loop to print the names of students who scored more than 75. Would you like to modify the list, change the score threshold, or try a different type of loop like a `for` loop?

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal

4:06 PM Python 3

Explanation:

Gemini X

K explanation

◆ Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 6ea779a8): This cell demonstrates how to use the `ClassName` defined in a previous cell.

- It creates an object named `my_object` from the `ClassName`, providing values for `attribute1`, `attribute2`, and an `initial_balance` of 100.
- It then calls the `check_balance()` method to display the initial balance.
- It calls the `deposit(50)` method to add 50 to the balance and shows the updated balance.
- It calls the `withdraw(30)` method to subtract 30 from the balance and shows the updated balance.
- It attempts to call `withdraw(200)` to demonstrate the "Insufficient funds" message because the current balance is less than 200.
- Finally, it calls `check_balance()` again to show the final balance after the operations.

Cell 2 (id: 1780c7f1): This cell defines a list of dictionaries, where each dictionary represents a student with a "name" and a "score".

- It then initializes a variable `i` to 0, which will be used as an index for the `while` loop.
- The `while i < len(students):` line starts a loop that will continue as long as the value of `i` is less than the total number of students in the list.
- Inside the loop, `student = students[i]` accesses the dictionary at the current index `i` and assigns it to the `student` variable.
- The `if student["score"] > 75:` line checks if the "score" of the current `student` is greater than 75.
- If the score is greater than 75, `print(student["name"])` prints the "name" of that student.
- Finally, `i += 1` increments the index `i` so that the loop moves to the next student in the list.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:06 PM Python 3

Task 5:

Begin writing a class ShoppingCart with an empty items list to generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.

Code and Output:

AI(6).ipynb

File Edit View Insert Runtime Tools Help

Gemini X

Commands + Code + Text Run all

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

 def add_item(self, item):
 self.items.append(item)

 # Implement remove item method

Subtask:
Add a method to the ShoppingCart class that takes an item's name and removes it from the items list. Handle cases where the item is not found.

Reasoning: Add the `remove_item` method to the `ShoppingCart` class as instructed.

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

 def add_item(self, item):
 self.items.append(item)

 def remove_item(self, item_name):
 found = False
 for item in self.items:
 if item['name'] == item_name:
 self.items.remove(item)
 print(f'{item_name} removed from the shopping cart.')
 found = True
 break

Generating...
Preparing
Define the 'shoppingcart' class
Implement 'add item' method
Implement 'remove item' method
Implement 'calculate total' method
◆ What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal

4:19 PM Python 3

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[29] if found:
    print(f"{item_name} not found in the shopping cart.")

<> Implement calculate total method

Subtask:
```

Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.

Reasoning: Implement the `calculate_total` method in the `ShoppingCart` class as described in the instructions, including iterating through items, calculating subtotals, applying conditional discounts, and summing up the total.

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

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item_name):
        found = False
        for item in self.items:
            if item['name'] == item_name:
                self.items.remove(item)
                print(f"{item_name} removed from the shopping cart.")
                found = True
            break
        if not found:
            print(f"{item_name} not found in the shopping cart.")

    def calculate_total(self):
        total = 0
        for item in self.items:
            subtotal = item['price'] * item['quantity']
            # Apply conditional discounts
            if item['quantity'] > 5: # Example: 10% discount for quantity > 5
                discount = subtotal * 0.10
                subtotal -= discount
                print(f"Applied 10% discount to {item['name']} for quantity over 5.")
            elif item['price'] > 50: # Example: 5% discount for price over 50
                discount = subtotal * 0.05
                subtotal -= discount
                print(f"Applied 5% discount to {item['name']} for price over 50.")

            total += subtotal
        return total
```

Variables Terminal

Gemini RAM Disk

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:19 PM Python 3

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[29] if item['name'] == item_name:
    self.items.remove(item)
    print(f"{item_name} removed from the shopping cart.")
    found = True
    break
if not found:
    print(f"{item_name} not found in the shopping cart.")

def calculate_total(self):
    total = 0
    for item in self.items:
        subtotal = item['price'] * item['quantity']
        # Apply conditional discounts
        if item['quantity'] > 5: # Example: 10% discount for quantity > 5
            discount = subtotal * 0.10
            subtotal -= discount
            print(f"Applied 10% discount to {item['name']} for quantity over 5.")
        elif item['price'] > 50: # Example: 5% discount for price over 50
            discount = subtotal * 0.05
            subtotal -= discount
            print(f"Applied 5% discount to {item['name']} for price over 50.")

        total += subtotal
    return total
```

<> Add example usage

Subtask:

Variables Terminal

Gemini RAM Disk

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:19 PM Python 3

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[29] # Create an instance of the ShoppingCart class
my_cart = ShoppingCart()

# Create item dictionaries
item1 = {'name': 'Laptop', 'price': 1000, 'quantity': 1}
item2 = {'name': 'Mouse', 'price': 25, 'quantity': 10}
item3 = {'name': 'Keyboard', 'price': 75, 'quantity': 2}
item4 = {'name': 'Monitor', 'price': 300, 'quantity': 1}

# Add items to the shopping cart
my_cart.add_item(item1)
my_cart.add_item(item2)
my_cart.add_item(item3)
my_cart.add_item(item4)

# Remove an item
my_cart.remove_item('Mouse')

# Calculate the total
total_bill = my_cart.calculate_total()

# Print the total
print(f"The total bill is: ${total_bill:.2f}")

# Mouse removed from the shopping cart.
Applied 5% discount to Laptop for price over 50.
Applied 5% discount to Keyboard for price over 50.
Applied 5% discount to Monitor for price over 50.
The total bill is: $1377.50
```

Variables Terminal

Gemini RAM Disk

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:19 PM Python 3

Explanation:

Gemini X

...

K Explanation

← Certainly, I can explain the content of the cells you have visible.

Cell 1 (id: ccb80eed): This is a Markdown cell that introduces the subtask of adding example usage to the `ShoppingCart` class. It explains that the goal is to create an instance of the class, add and remove items, and call the `calculate_total` method to show how it works.

Cell 2 (id: e4ec64e6): This is a Markdown cell providing the reasoning for the next code cell. It states that the next step is to create an instance of the `ShoppingCart` class, define item dictionaries, add and remove items, calculate the total, and print the result.

Cell 3 (id: 2d979ea4): This is a code cell that demonstrates the usage of the `ShoppingCart` class.

- It creates an instance of the `ShoppingCart` class called `my_cart`.
- It defines four dictionaries (`item1`, `item2`, `item3`, `item4`) representing different items with their names, prices, and quantities.
- It adds all four items to the `my_cart` using the `add_item` method.
- It then removes the item named 'Mouse' from the cart using the `remove_item` method.
- It calls the `calculate_total()` method on `my_cart` to calculate the total bill after adding and removing items, and stores the result in `total_bill`.
- Finally, it prints the `total_bill` formatted as a currency value with two decimal places.

Cell 4 (id: d38aac2b): This is a Markdown cell that provides a summary of the task, key findings, and insights or next steps.

- It summarizes that the `ShoppingCart` class was successfully created with methods for adding, removing, and calculating the total with discounts.
- It lists key findings, such as the successful implementation of each method and the details of the conditional discounts applied (10% for quantity > 5 and 5% for price > \$50).
- It suggests insights or next steps, including adding validation to the `add_item` method and considering more complex discount rules.

◆ What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)