

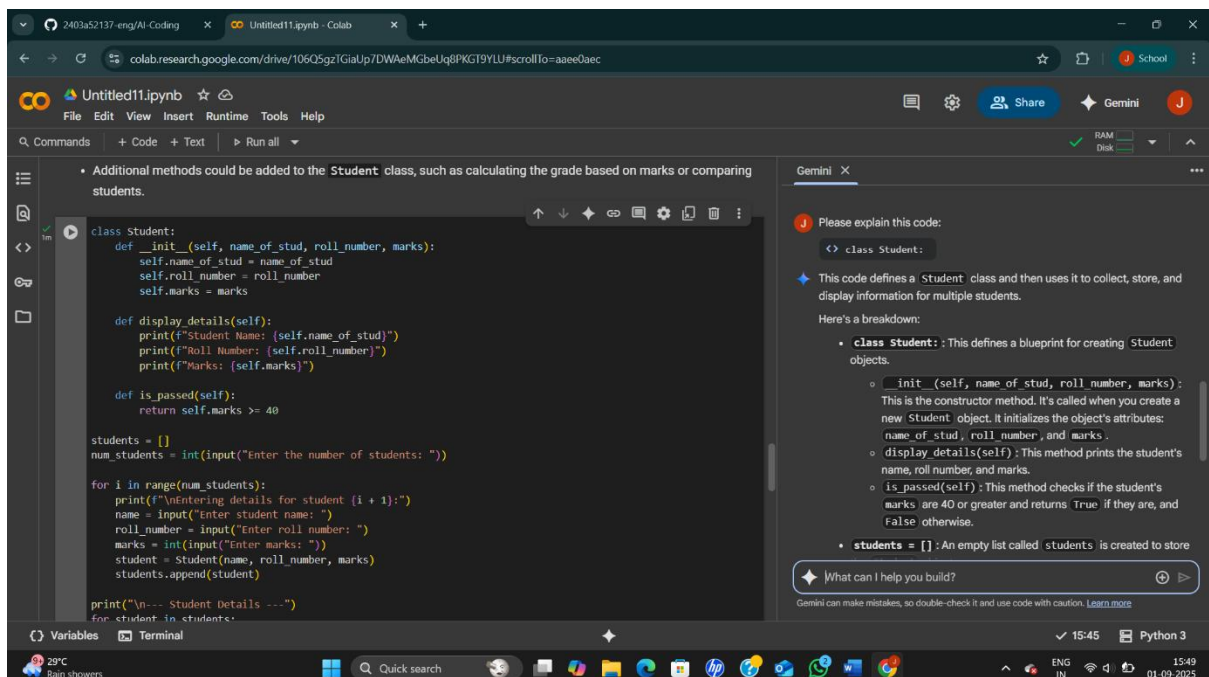
AI ASSISTED CODING

LAB 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 SCREENSHOTS:



The screenshot shows a Google Colab notebook titled 'Untitled11.ipynb'. The code defines a `Student` class with attributes `name`, `roll_number`, and `marks`. It includes methods `display_details` and `is_passed`. The `is_passed` method checks if marks are greater than or equal to 40. The notebook also shows a list `students` and a loop to collect student data.

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

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

    def is_passed(self):
        return self.marks >= 40

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

for i in range(num_students):
    print(f"Entering details for student {i + 1}:")
    name = input("Enter student name: ")
    roll_number = input("Enter roll number: ")
    marks = int(input("Enter marks: "))
    student = Student(name, roll_number, marks)
    students.append(student)

print("\n--- Student Details ---")
for student in students:
```

The Gemini AI assistant explains the code:

- class Student:** This defines a blueprint for creating Student objects.
- __init__(self, name_of_stud, roll_number, marks):** This is the constructor method. It's called when you create a new Student object. It initializes the object's attributes: name_of_stud, roll_number, and marks.
- display_details(self):** This method prints the student's name, roll number, and marks.
- is_passed(self):** This method checks if the student's marks are 40 or greater and returns True if they are, and False otherwise.
- students = []:** An empty list called students is created to store objects.

What can I help you build?

2403a52137-eng/AI-Coding x Untitled11.ipynb - Colab

colab.research.google.com/drive/106Q5gz7GiaUp7DWAeMGbeUq8PKG19YLU#scrollTo=aaee0aec

Untitled11.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
student = Student(name, roll_number, marks)
students.append(student)

print("\n--- Student Details ---")
for student in students:
    student.display_details()
    if student.is_passed():
        print("Status: Passed")
    else:
        print("Status: Failed")
    print("-" * 20)

Enter the number of students: 5

Entering details for student 1:
Enter student name: nandhu
Enter roll number: 2086
Enter marks: 89

Entering details for student 2:
Enter student name: bhavitha
Enter roll number: 2099
Enter marks: 92

Entering details for student 3:
Enter student name: akshitha
Enter roll number: 2137
Enter marks: 45

Entering details for student 4:
Enter student name: jyoshna
Enter roll number: 2106
Enter marks: 55
```

Gemini

- `is_passed(self)`: This method checks if the student's marks are 40 or greater and returns `True` if they are, and `False` otherwise.
- `students = []`: An empty list called `students` is created to store the `Student` objects.
- `num_students = int(input("Enter the number of students: "))`: The program prompts the user to enter the number of students and stores the input as an integer in the `num_students` variable.
- `for i in range(num_students): ...`: This loop runs `num_students` times to get details for each student.
 - Inside the loop, it prompts the user for the student's name, roll number, and marks.
 - `student = Student(name, roll_number, marks)`: A new `Student` object is created with the entered details.
 - `students.append(student)`: The newly created `Student` object is added to the `students` list.
- `print("\n--- Student Details ---")`: This prints a header before displaying the student details.
- `for student in students: ...`: This loop iterates through each `Student` object in the `students` list.

What can I help you build?

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

15:45 Python 3

29°C Mostly cloudy

Quick search

01-09-2025

2403a52137-eng/AI-Coding x Untitled11.ipynb - Colab

colab.research.google.com/drive/106Q5gz7GiaUp7DWAeMGbeUq8PKG19YLU#scrollTo=aaee0aec

Untitled11.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
Enter the number of students: 5

Entering details for student 1:
Enter student name: nandhu
Enter roll number: 2086
Enter marks: 89

Entering details for student 2:
Enter student name: bhavitha
Enter roll number: 2099
Enter marks: 92

Entering details for student 3:
Enter student name: akshitha
Enter roll number: 2137
Enter marks: 45

Entering details for student 4:
Enter student name: jyoshna
Enter roll number: 2106
Enter marks: 55

--- Student Details ---
Student Name: nandhu
Roll Number: 2086
Marks: 89
Status: Passed
-----
Student Name: bhavitha
Roll Number: 2099
Marks: 92
Status: Passed
-----
Student Name: akshitha
Roll Number: 2137
Marks: 45
Status: Passed
-----
Student Name: jyoshna
Roll Number: 2106
Marks: 55
Status: Passed
-----
Student Name: sreeja
Roll Number: 2103
Marks: 67
```

Gemini

- `num_students` times to get details for each student.
- Inside the loop, it prompts the user for the student's name, roll number, and marks.
- `student = Student(name, roll_number, marks)`: A new `Student` object is created with the entered details.
- `students.append(student)`: The newly created `Student` object is added to the `students` list.
- `print("\n--- Student Details ---")`: This prints a header before displaying the student details.
- `for student in students: ...`: This loop iterates through each `Student` object in the `students` list.
 - `student.display_details()`: Calls the `display_details` method for the current student to print their information.
 - `if student.is_passed(): ... else: ...`: This checks if the student passed using the `is_passed` method and prints "Status: Passed" or "Status: Failed" accordingly.
 - `print("-" * 20)`: Prints a separator line after each student's details.

Bad response

What can I help you build?

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

15:45 Python 3

Breaking news PM Modi at SCO...

Quick search

15:52

01-09-2025

CODE EXPLANATION:

Gemini

✕

...

J

Please explain this code:

<>

class Student:

✦

This code defines a `Student` class and then uses it to collect, store, and display information for multiple students.

Here's a breakdown:

- `class Student`: This defines a blueprint for creating `Student` objects.
 - `__init__(self, name_of_stud, roll_number, marks)`: This is the constructor method. It's called when you create a new `Student` object. It initializes the object's attributes: `name_of_stud`, `roll_number`, and `marks`.
 - `display_details(self)`: This method prints the student's name, roll number, and marks.
 - `is_passed(self)`: This method checks if the student's `marks` are 40 or greater and returns `True` if they are, and `False` otherwise.
- `students = []`: An empty list called `students` is created to store

✦

What can I help you build?

+

➤

Gemini

✕

...

- `is_passed(self)`: This method checks if the student's `marks` are 40 or greater and returns `True` if they are, and `False` otherwise.
- `students = []`: An empty list called `students` is created to store the `Student` objects.
- `num_students = int(input("Enter the number of students: "))`: The program prompts the user to enter the number of students and stores the input as an integer in the `num_students` variable.
- `for i in range(num_students): ...`: This loop runs `num_students` times to get details for each student.
 - Inside the loop, it prompts the user for the student's name, roll number, and marks.
 - `student = Student(name, roll_number, marks)`: A new `Student` object is created with the entered details.
 - `students.append(student)`: The newly created `Student` object is added to the `students` list.
- `print("\n--- Student Details ---")`: This prints a header before displaying the student details.
- `for student in students: ...`: This loop iterates through each `Student` object in the `students` list.

✦

What can I help you build?

+

➤

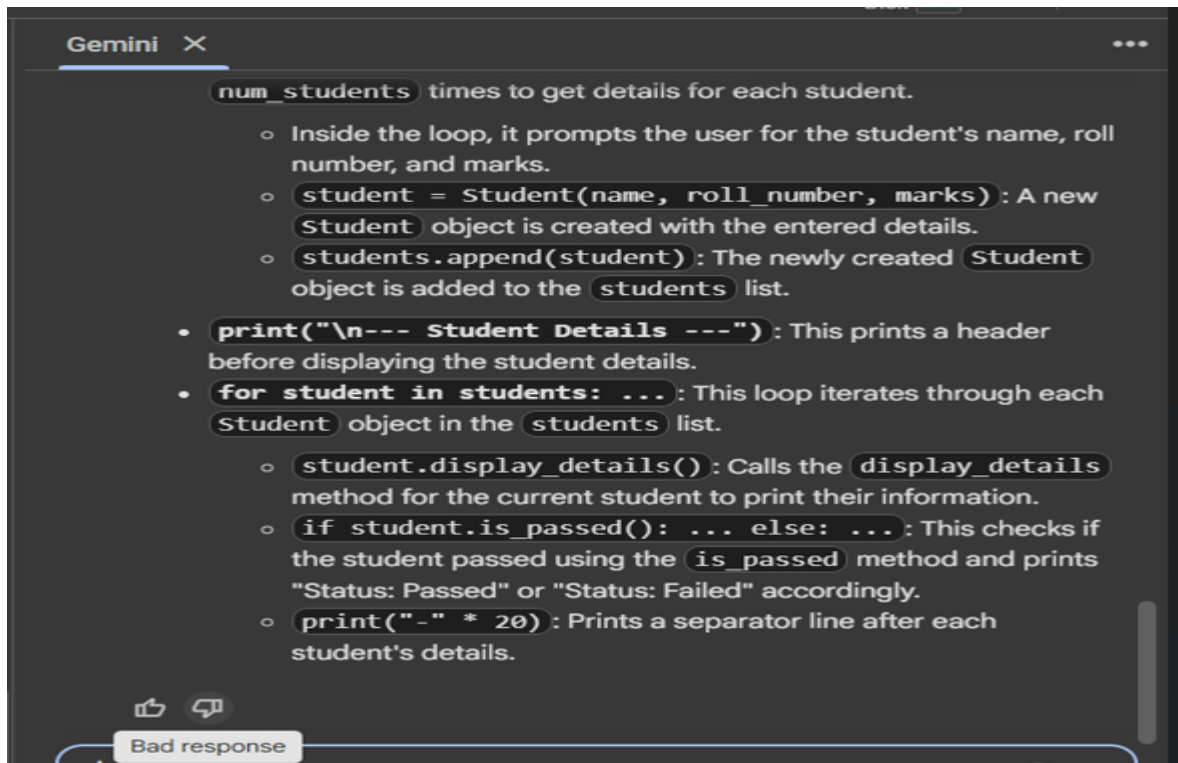
Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

✓ 15:45

Python 3

15:51

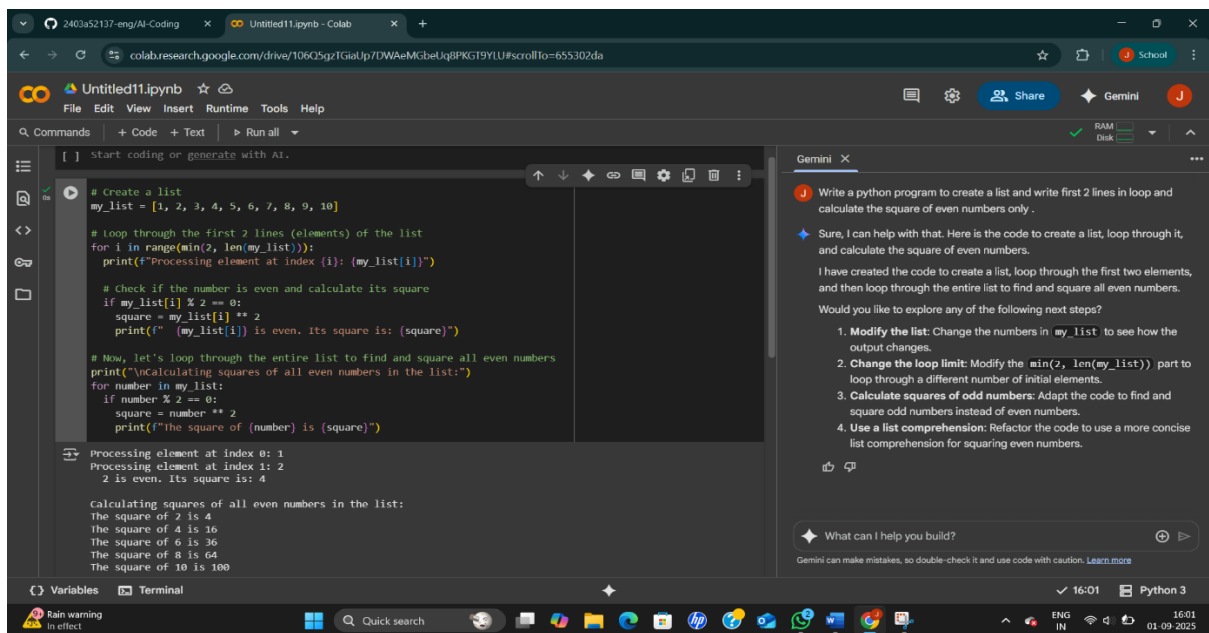
01-09-2025



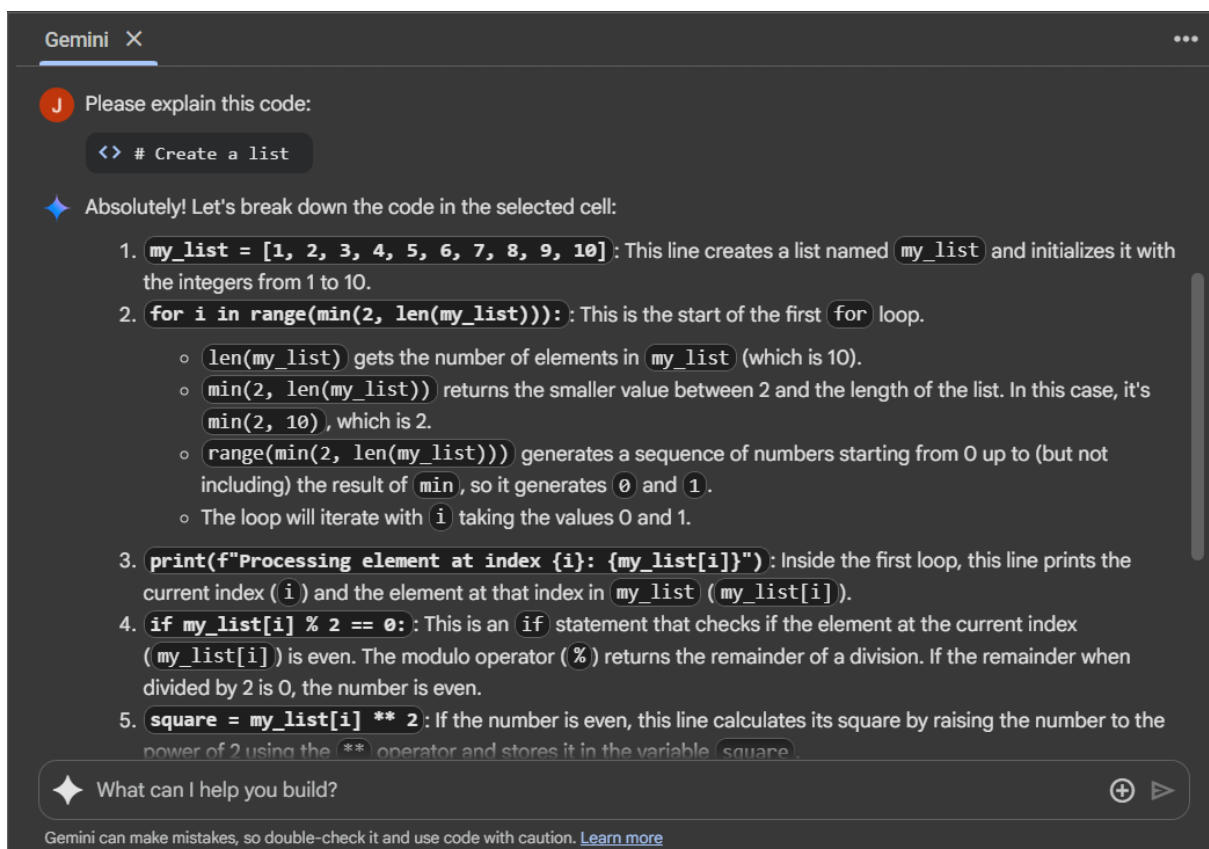
TASK-2:

- 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:



CODE EXPLANATION:



Gemini ✕

3. `print(f"Processing element at index {i}: {my_list[i]}")` : Inside the first loop, this line prints the current index (`i`) and the element at that index in `my_list` (`my_list[i]`).

4. `if my_list[i] % 2 == 0:` : This is an `if` statement that checks if the element at the current index (`my_list[i]`) is even. The modulo operator (`%`) returns the remainder of a division. If the remainder when divided by 2 is 0, the number is even.

5. `square = my_list[i] ** 2` : If the number is even, this line calculates its square by raising the number to the power of 2 using the `**` operator and stores it in the variable `square`.

6. `print(f" {my_list[i]} is even. Its square is: {square}")` : This line prints a message indicating that the number is even and displays its calculated square.

7. `print("\nCalculating squares of all even numbers in the list:")` : After the first loop finishes, this line prints a blank line (`\n`) and a descriptive message.

8. `for number in my_list:` : This is the start of the second `for` loop. This loop iterates directly over each `number` in the `my_list`.

9. `if number % 2 == 0:` : Similar to the first loop, this checks if the current `number` is even.

10. `square = number ** 2` : If the number is even, its square is calculated and stored in the `square` variable.

11. `print(f"The square of {number} is {square}")` : This line prints the original even number and its calculated square.

In summary, the code first processes only the first two elements of the list, checking if they are even and printing their squares if they are. Then, it goes through the *entire* list again to find and print the squares of *all* the even numbers.

👍

🔗

✦

What can I help you build?

⊕

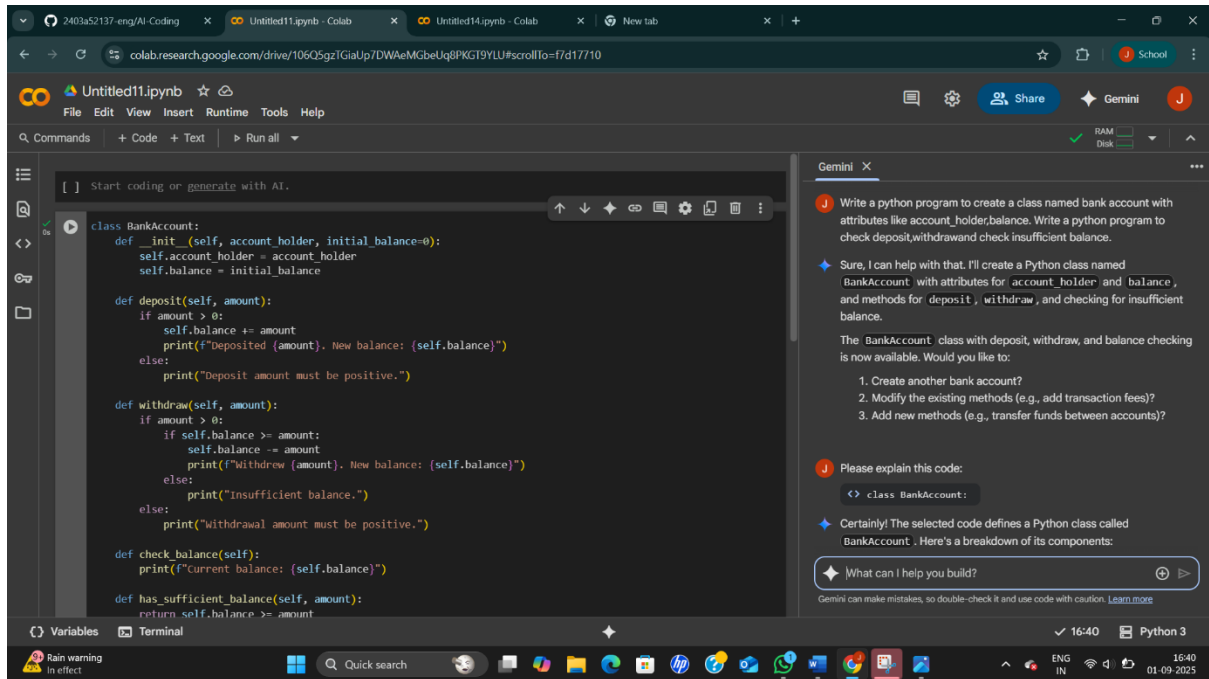
➤

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

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 :



The screenshot shows a Google Colab notebook with the following code in the cell:

```
[ ] Start coding or generate with AI.  
  
class BankAccount:  
    def __init__(self, account_holder, initial_balance=0):  
        self.account_holder = account_holder  
        self.balance = initial_balance  
  
    def deposit(self, amount):  
        if amount > 0:  
            self.balance += amount  
            print(f"Deposited {amount}. 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"Withdrew {amount}. New balance: {self.balance}")  
            else:  
                print("Insufficient balance.")  
        else:  
            print("Withdrawal amount must be positive.")  
  
    def check_balance(self):  
        print(f"Current balance: {self.balance}")  
  
    def has_sufficient_balance(self, amount):  
        return self.balance >= amount
```

The Gemini chat interface on the right shows the following conversation:

J Write a python program to create a class named bank account with attributes like account_holder,balance. Write a python program to check deposit,withdrawand check insufficient balance.

◆ Sure, I can help with that. I'll create a Python class named `BankAccount` with attributes for `account_holder` and `balance`, and methods for `deposit`, `withdraw`, and checking for insufficient balance.

The `BankAccount` class with `deposit`, `withdraw`, and `balance` checking is now available. Would you like to:

1. Create another bank account?
2. Modify the existing methods (e.g., add transaction fees)?
3. Add new methods (e.g., transfer funds between accounts)?

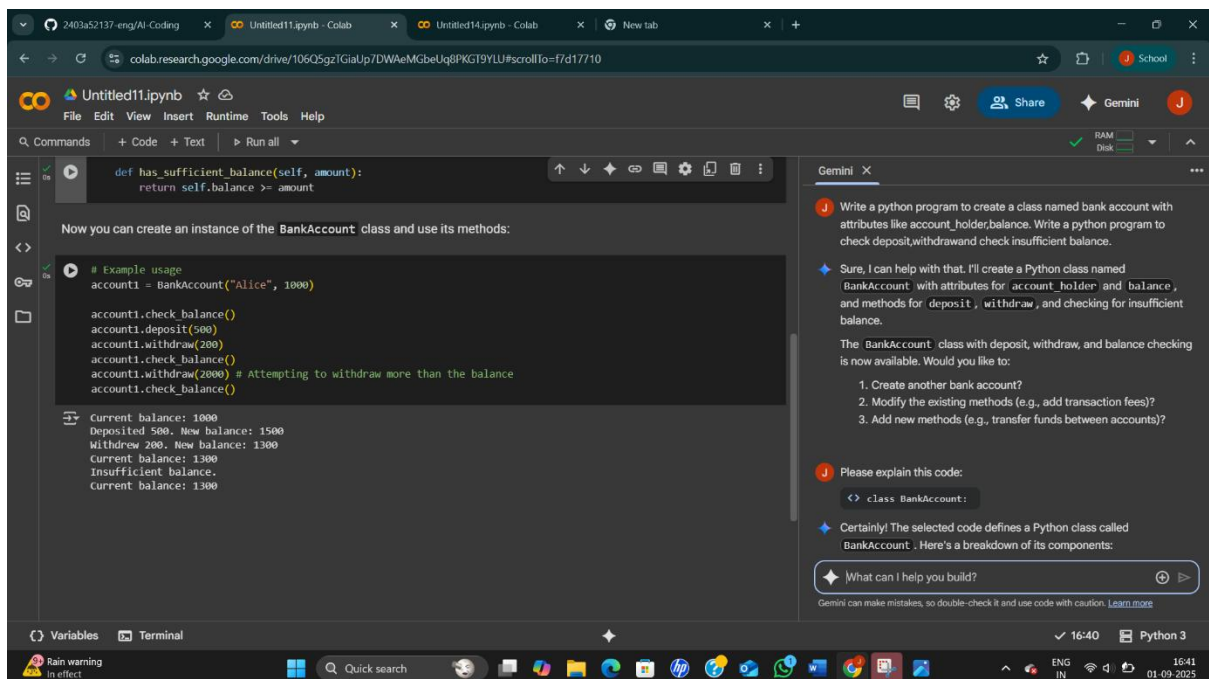
J Please explain this code:

◆ `class BankAccount:`

◆ Certainly! The selected code defines a Python class called `BankAccount`. Here's a breakdown of its components:

◆ What can I help you build?

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



The screenshot shows the same Google Colab notebook with the following code in the cell:

```
def has_sufficient_balance(self, amount):  
    return self.balance >= amount
```

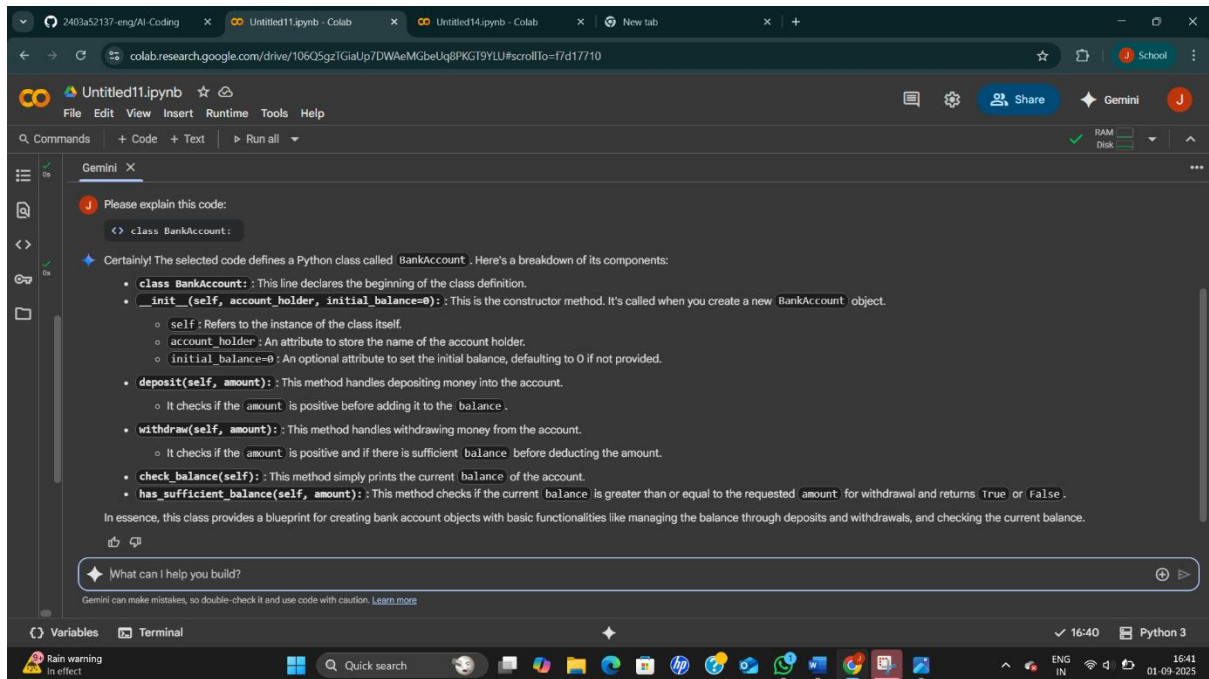
Now you can create an instance of the `BankAccount` class and use its methods:

```
# Example usage  
account1 = BankAccount("Alice", 1000)  
  
account1.check_balance()  
account1.deposit(500)  
account1.withdraw(200)  
account1.check_balance()  
account1.withdraw(2000) # Attempting to withdraw more than the balance  
account1.check_balance()
```

Current balance: 1000
Deposited 500. New balance: 1500
Withdrew 200. New balance: 1300
Current balance: 1300
Insufficient balance.
Current balance: 1300

The Gemini chat interface on the right shows the same conversation as the first screenshot.

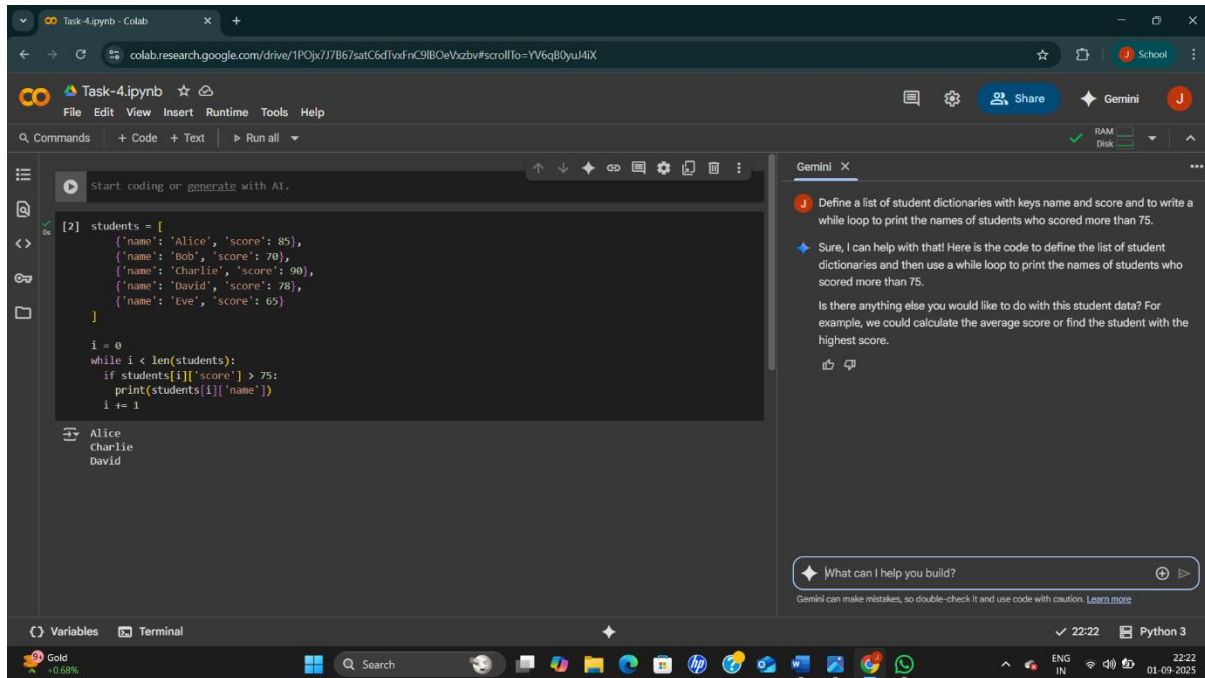
CODE EXPLANATION:



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 :



The screenshot shows a Google Colab notebook titled 'Task-4.ipynb'. The code in the notebook is as follows:

```
[2] students = [
    {'name': 'Alice', 'score': 85},
    {'name': 'Bob', 'score': 70},
    {'name': 'Charlie', 'score': 90},
    {'name': 'David', 'score': 78},
    {'name': 'Eve', 'score': 65}
]

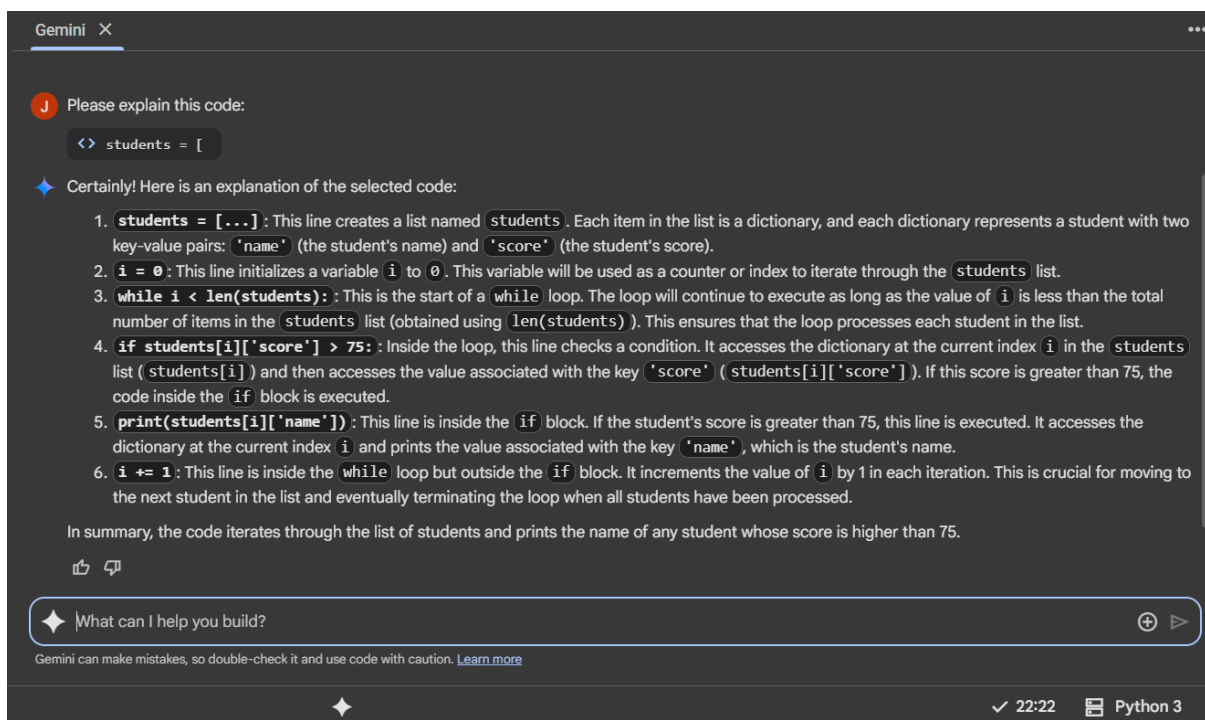
i = 0
while i < len(students):
    if students[i]['score'] > 75:
        print(students[i]['name'])
    i += 1
```

The output of the code is:

```
Alice
Charlie
David
```

On the right side, the Gemini AI chat interface is visible. It shows a prompt: '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.' The AI responds with a helpful message and offers to calculate the average score or find the student with the highest score.

CODE EXPLANATION:



The screenshot shows the Gemini AI chat interface. The user prompt is: 'Please explain this code:'. The AI response is:

Certainly! Here is an explanation of the selected code:

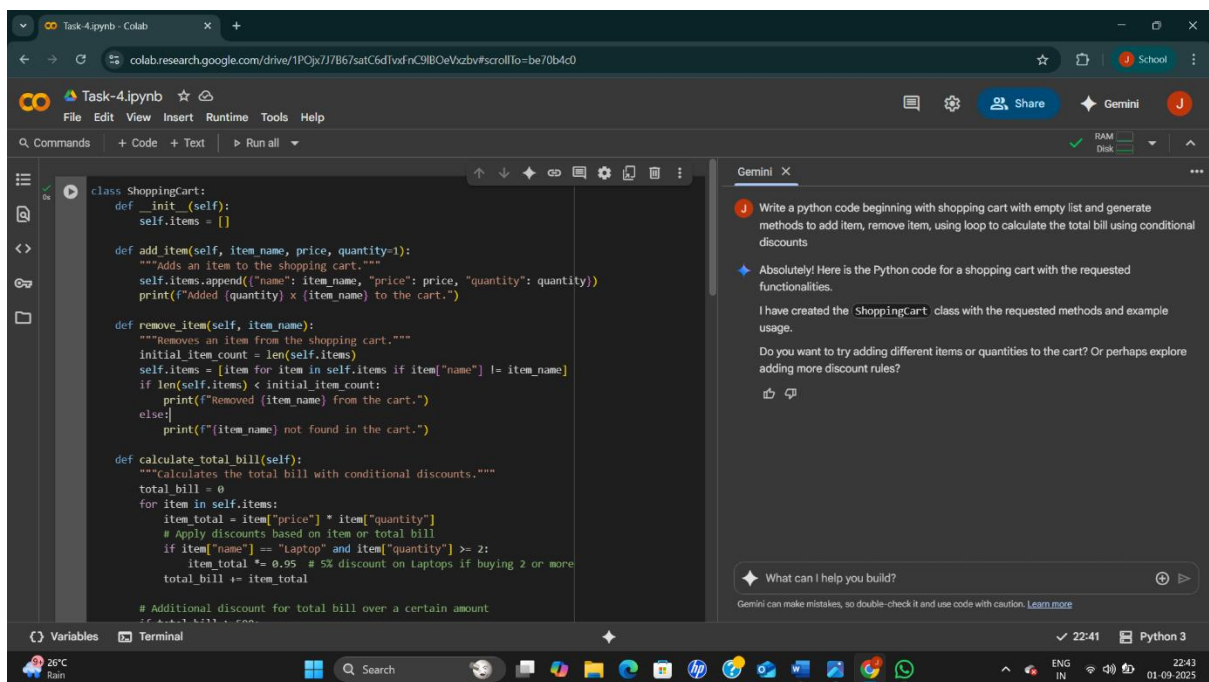
- students = [...]**: This line creates a list named `students`. Each item in the list is a dictionary, and each dictionary represents a student with two key-value pairs: `'name'` (the student's name) and `'score'` (the student's score).
- i = 0**: This line initializes a variable `i` to 0. This variable will be used as a counter or index to iterate through the `students` list.
- while i < len(students):**: This is the start of a `while` loop. The loop will continue to execute as long as the value of `i` is less than the total number of items in the `students` list (obtained using `len(students)`). This ensures that the loop processes each student in the list.
- if students[i]['score'] > 75:**: Inside the loop, this line checks a condition. It accesses the dictionary at the current index `i` in the `students` list (`students[i]`) and then accesses the value associated with the key `'score'` (`students[i]['score']`). If this score is greater than 75, the code inside the `if` block is executed.
- print(students[i]['name'])**: This line is inside the `if` block. If the student's score is greater than 75, this line is executed. It accesses the dictionary at the current index `i` and prints the value associated with the key `'name'`, which is the student's name.
- i += 1**: This line is inside the `while` loop but outside the `if` block. It increments the value of `i` by 1 in each iteration. This is crucial for moving to the next student in the list and eventually terminating the loop when all students have been processed.

In summary, the code iterates through the list of students and prints the name of any student whose score is higher than 75.

TASK-5:

Write a python code beginning with shopping cart with empty list and generate methods to add item, remove item, using loop to calculate the total bill using conditional discounts.

CODE:



The screenshot shows a Google Colab notebook titled 'Task-4.ipynb' with a Python class named `ShoppingCart`. The class has three methods: `__init__` (initializes an empty list `self.items`), `add_item` (adds an item to the list with name, price, and quantity), and `remove_item` (removes an item from the list). A `calculate_total_bill` method is also present, which uses a loop to calculate the total bill with conditional discounts. The Gemini chat interface on the right shows a conversation about the code.

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

    def add_item(self, item_name, price, quantity=1):
        """Adds an item to the shopping cart."""
        self.items.append({"name": item_name, "price": price, "quantity": quantity})
        print(f"Added {quantity} x {item_name} to the cart.")

    def remove_item(self, item_name):
        """Removes an item from the shopping cart."""
        initial_item_count = len(self.items)
        self.items = [item for item in self.items if item["name"] != item_name]
        if len(self.items) < initial_item_count:
            print(f"Removed {item_name} from the cart.")
        else:
            print(f"{item_name} not found in the cart.")

    def calculate_total_bill(self):
        """Calculates the total bill with conditional discounts."""
        total_bill = 0
        for item in self.items:
            item_total = item["price"] * item["quantity"]
            # Apply discounts based on item or total bill
            if item["name"] == "Laptop" and item["quantity"] >= 2:
                item_total *= 0.95 # 5% discount on Laptops if buying 2 or more
            total_bill += item_total

        # Additional discount for total bill over a certain amount
        if total_bill > 100:
            total_bill *= 0.9
```

Gemini

Write a python code beginning with shopping cart with empty list and generate methods to add item, remove item, using loop to calculate the total bill using conditional discounts

Absolutely! Here is the Python code for a shopping cart with the requested functionalities.

I have created the `ShoppingCart` class with the requested methods and example usage.

Do you want to try adding different items or quantities to the cart? Or perhaps explore adding more discount rules?

What can I help you build?

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

The screenshot shows a Google Colab environment. The main notebook area contains Python code for a shopping cart class. The code includes a 10% discount on the total bill if it exceeds \$500. The Gemini chat interface on the right provides a prompt and a response.

```
if total_bill > 500:
    total_bill *= 0.90 # 10% discount on total bill if over $500

return total_bill

def display_cart(self):
    """Displays the current items in the shopping cart."""
    if not self.items:
        print("Your shopping cart is empty.")
    else:
        print("Items in your shopping cart:")
        for item in self.items:
            print(f"- {item['name']} (Price: ${item['price']:.2f}, Quantity: {item['quantity']})")

# Example usage:
cart = ShoppingCart()
cart.add_item("Laptop", 1200, 2)
cart.add_item("Mouse", 25)
cart.add_item("Keyboard", 75)
cart.add_item("Laptop", 1200) # Adding another laptop

cart.display_cart()

total = cart.calculate_total_bill()
print(f"\nTotal bill: ${total:.2f}")

cart.remove_item("Mouse")
cart.display_cart()

total = cart.calculate_total_bill()
print(f"\nTotal bill after removing Mouse: ${total:.2f}")
```

Gemini Chat:

J Write a python code beginning with shopping cart with empty list and generate methods to add item, remove item, using loop to calculate the total bill using conditional discounts

+ Absolutely! Here is the Python code for a shopping cart with the requested functionalities.

I have created the `ShoppingCart` class with the requested methods and example usage.

Do you want to try adding different items or quantities to the cart? Or perhaps explore adding more discount rules?

What can I help you build?

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

OUTPUT:

The output of the code execution is displayed in a dark-themed window. It shows the step-by-step process of adding items to the cart, displaying the current state, removing an item, and calculating the final bill with a discount.

```
Added 2 x Laptop to the cart.
Added 1 x Mouse to the cart.
Added 1 x Keyboard to the cart.
Added 1 x Laptop to the cart.
Items in your shopping cart:
- Laptop (Price: $1200.00, Quantity: 2)
- Mouse (Price: $25.00, Quantity: 1)
- Keyboard (Price: $75.00, Quantity: 1)
- Laptop (Price: $1200.00, Quantity: 1)

Total bill: $3222.00
Removed Mouse from the cart.
Items in your shopping cart:
- Laptop (Price: $1200.00, Quantity: 2)
- Keyboard (Price: $75.00, Quantity: 1)
- Laptop (Price: $1200.00, Quantity: 1)

Total bill after removing Mouse: $3199.50
```

CODE EXPLANATION:

Gemini

Please explain this code:

<> class ShoppingCart:

Certainly! The code defines a `ShoppingCart` class in Python. Let's break down its components:

- `__init__(self)` : This is the constructor of the class. When you create a new `ShoppingCart` object, this method is automatically called. It initializes an empty list called `self.items` which will store the items in the cart.
- `add_item(self, item_name, price, quantity=1)` : This method adds an item to the `self.items` list. Each item is stored as a dictionary containing its "name", "price", and "quantity". It also prints a confirmation message.
- `remove_item(self, item_name)` : This method removes an item from the `self.items` list based on its name. It uses a list comprehension to create a new list excluding the item to be removed. It then prints a message indicating whether the item was removed or not found.
- `calculate_total_bill(self)` : This method calculates the total cost of all items in the cart. It iterates through the `self.items` list, calculates the total for each item (price * quantity), and applies conditional discounts:
 - A 5% discount is applied to "Laptop" items if the quantity is 2 or more.
 - A 10% discount is applied to the *total* bill if it exceeds \$500.
- `display_cart(self)` : This method prints the current contents of the shopping cart, listing each item with its price and quantity. If the cart is empty, it prints a message indicating that.

The code then demonstrates how to use the `ShoppingCart` class by creating an instance, adding and removing items, and calculating and displaying the total bill at different stages.

What can I help you build?

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

22:41

Python 3