

# AI-Assisted Coding

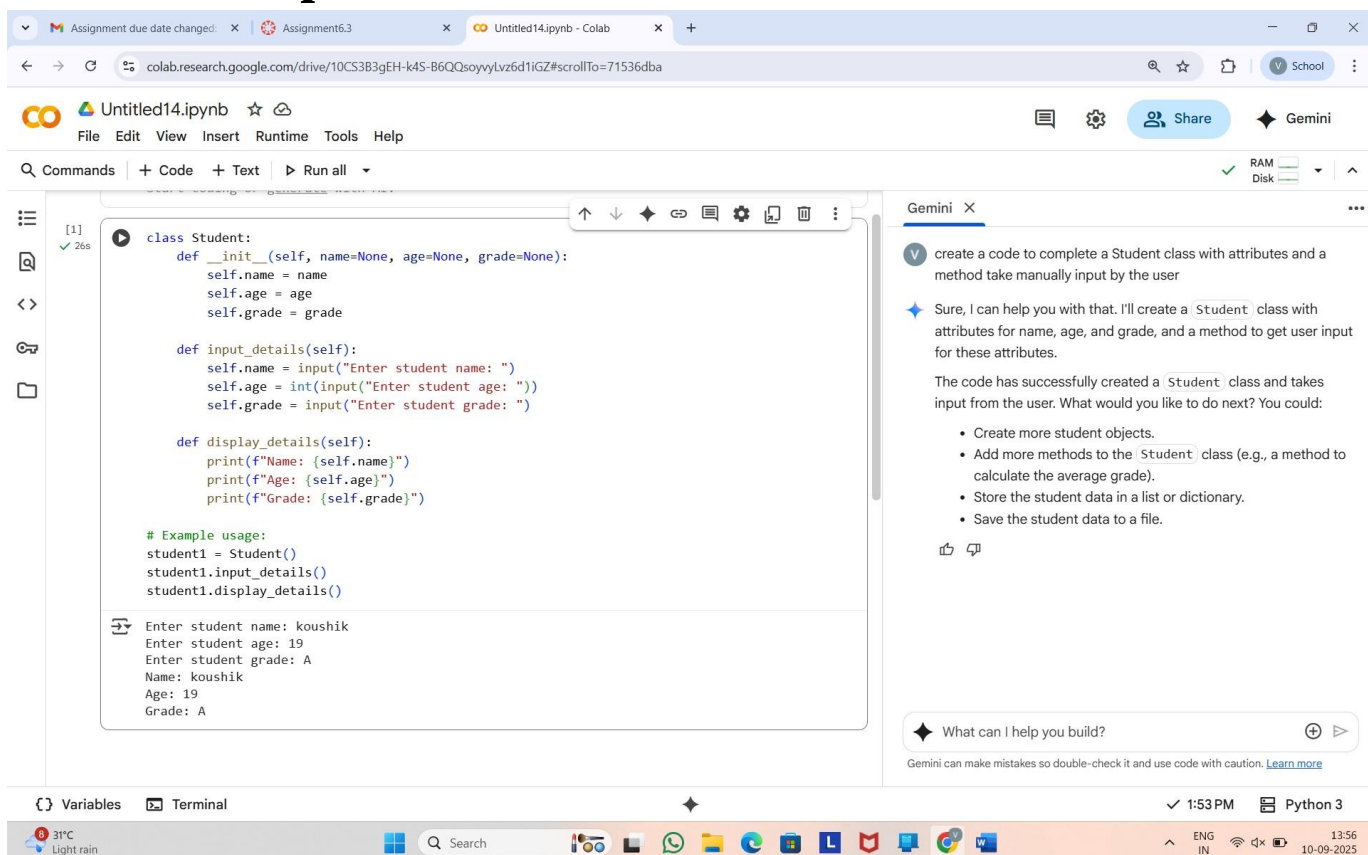
Name: Ashish Alla

Htno:2403A51249

## Task-1:

**Prompt:** Create a code to complete a Student class with attributes and a method take manually input by the user.

## Code&Output:



The screenshot displays a Google Colab notebook interface. The main editor shows a Python class named `Student` with the following code:

```
[1] ✓ 26s
class Student:
    def __init__(self, name=None, age=None, grade=None):
        self.name = name
        self.age = age
        self.grade = grade

    def input_details(self):
        self.name = input("Enter student name: ")
        self.age = int(input("Enter student age: "))
        self.grade = input("Enter student grade: ")

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")

# Example usage:
student1 = Student()
student1.input_details()
student1.display_details()
```

The output of the code execution is shown below the code cell:

```
Enter student name: koushik
Enter student age: 19
Enter student grade: A
Name: koushik
Age: 19
Grade: A
```

On the right side of the notebook, the Gemini chat interface is visible. It shows a prompt: "create a code to complete a Student class with attributes and a method take manually input by the user". The Gemini response is:

Sure, I can help you with that. I'll create a `Student` class with attributes for name, age, and grade, and a method to get user input for these attributes.

The code has successfully created a `Student` class and takes input from the user. What would you like to do next? You could:

- Create more student objects.
- Add more methods to the `Student` class (e.g., a method to calculate the average grade).
- Store the student data in a list or dictionary.
- Save the student data to a file.

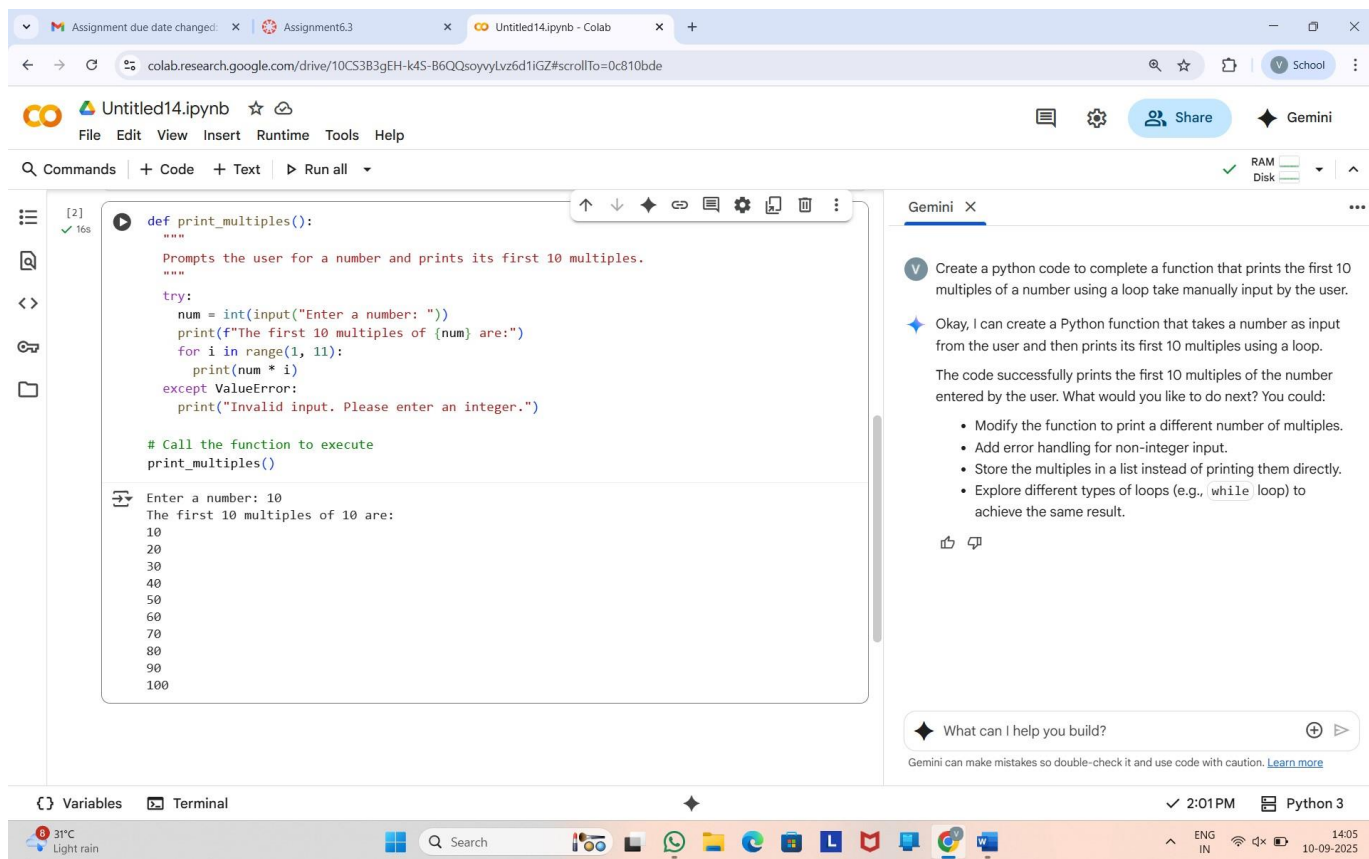
At the bottom of the Gemini chat, there is a prompt: "What can I help you build?".

**Observation:** The task successfully demonstrated how to create a Python class with attributes and a method that takes user input to populate those attributes. This is a fundamental concept in object-oriented programming for creating custom data structures and behaviors.

**Task-2:**

**Prompt:** Create a python code to complete a function that prints the first 10 multiples of a number using a loop take manually input by the user.

**Code&Output:**



**Observation:** This task successfully demonstrated how to define a Python function that takes user input and uses a for loop to iterate and perform a calculation (multiplication) a specific number of times. It also included basic error handling for invalid input

**Task-3:**

**Prompt:** Create a python code to write nested if-elifelse conditionals to classify age groups take mutiple manually input by the user.

**Code&Output:**

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyvyLvz6d1IGZ#scrollTo=1c2e3964

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[5] 21s

```
def classify_age_multiple():
    """
    Prompts the user for multiple ages and classifies them into age groups
    using nested if-elif-else conditionals.
    """
    while True:
        try:
            age_input = input("Enter age (or type 'quit' to exit): ")
            if age_input.lower() == 'quit':
                break
            age = int(age_input)

            if age < 0:
                print("Age cannot be negative.")
            elif age <= 12:
                print("You are a child.")
            else:
                if age <= 17:
                    print("You are a teenager.")
                else:
                    if age <= 64:
                        print("You are an adult.")
                    else:
                        print("You are a senior.")

        except ValueError:
            print("Invalid input. Please enter a valid integer for age or 'quit'.")

# Call the function to execute
classify_age_multiple()
```

Gemini

create a python code to write nested if-elif-else conditionals to classify age groups take multiple manually input by the user

I can modify the existing code to allow for multiple age inputs and classify each one using the nested if-elif-else conditionals.

The code now successfully takes multiple age inputs and classifies each one into age groups using nested if-elif-else statements.

What would you like to do next? You could:

- Store the classified ages and their categories.
- Count how many people fall into each age group.
- Validate the input further (e.g., ensure the input is within a reasonable age range).
- Integrate this age classification into a larger program or data analysis task.

What can I help you build?

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

Variables Terminal

31°C Light rain

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyvyLvz6d1IGZ#scrollTo=1c2e3964

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[5] 21s

```
classify_age_multiple()

Enter age (or type 'quit' to exit): 19
You are an adult.
Enter age (or type 'quit' to exit): 5
You are a child.
Enter age (or type 'quit' to exit): 17
You are a teenager.
Enter age (or type 'quit' to exit): quit
```

Gemini

create a python code to write nested if-elif-else conditionals to classify age groups take multiple manually input by the user

I can modify the existing code to allow for multiple age inputs and classify each one using the nested if-elif-else conditionals.

The code now successfully takes multiple age inputs and classifies each one into age groups using nested if-elif-else statements.

What would you like to do next? You could:

- Store the classified ages and their categories.
- Count how many people fall into each age group.
- Validate the input further (e.g., ensure the input is within a reasonable age range).
- Integrate this age classification into a larger program or data analysis task.

What can I help you build?

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

Variables Terminal

31°C Light rain

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyvyLvz6d1IGZ#scrollTo=1c2e3964

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[5] 21s

```
classify_age_multiple()

Enter age (or type 'quit' to exit): 19
You are an adult.
Enter age (or type 'quit' to exit): 5
You are a child.
Enter age (or type 'quit' to exit): 17
You are a teenager.
Enter age (or type 'quit' to exit): quit
```

Gemini

create a python code to write nested if-elif-else conditionals to classify age groups take multiple manually input by the user

I can modify the existing code to allow for multiple age inputs and classify each one using the nested if-elif-else conditionals.

The code now successfully takes multiple age inputs and classifies each one into age groups using nested if-elif-else statements.

What would you like to do next? You could:

- Store the classified ages and their categories.
- Count how many people fall into each age group.
- Validate the input further (e.g., ensure the input is within a reasonable age range).
- Integrate this age classification into a larger program or data analysis task.

What can I help you build?

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

Variables Terminal

31°C Light rain

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyvyLvz6d1IGZ#scrollTo=1c2e3964

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[5] 21s

```
classify_age_multiple()

Enter age (or type 'quit' to exit): 19
You are an adult.
Enter age (or type 'quit' to exit): 5
You are a child.
Enter age (or type 'quit' to exit): 17
You are a teenager.
Enter age (or type 'quit' to exit): quit
```

Gemini

create a python code to write nested if-elif-else conditionals to classify age groups take multiple manually input by the user

I can modify the existing code to allow for multiple age inputs and classify each one using the nested if-elif-else conditionals.

The code now successfully takes multiple age inputs and classifies each one into age groups using nested if-elif-else statements.

What would you like to do next? You could:

- Store the classified ages and their categories.
- Count how many people fall into each age group.
- Validate the input further (e.g., ensure the input is within a reasonable age range).
- Integrate this age classification into a larger program or data analysis task.

What can I help you build?

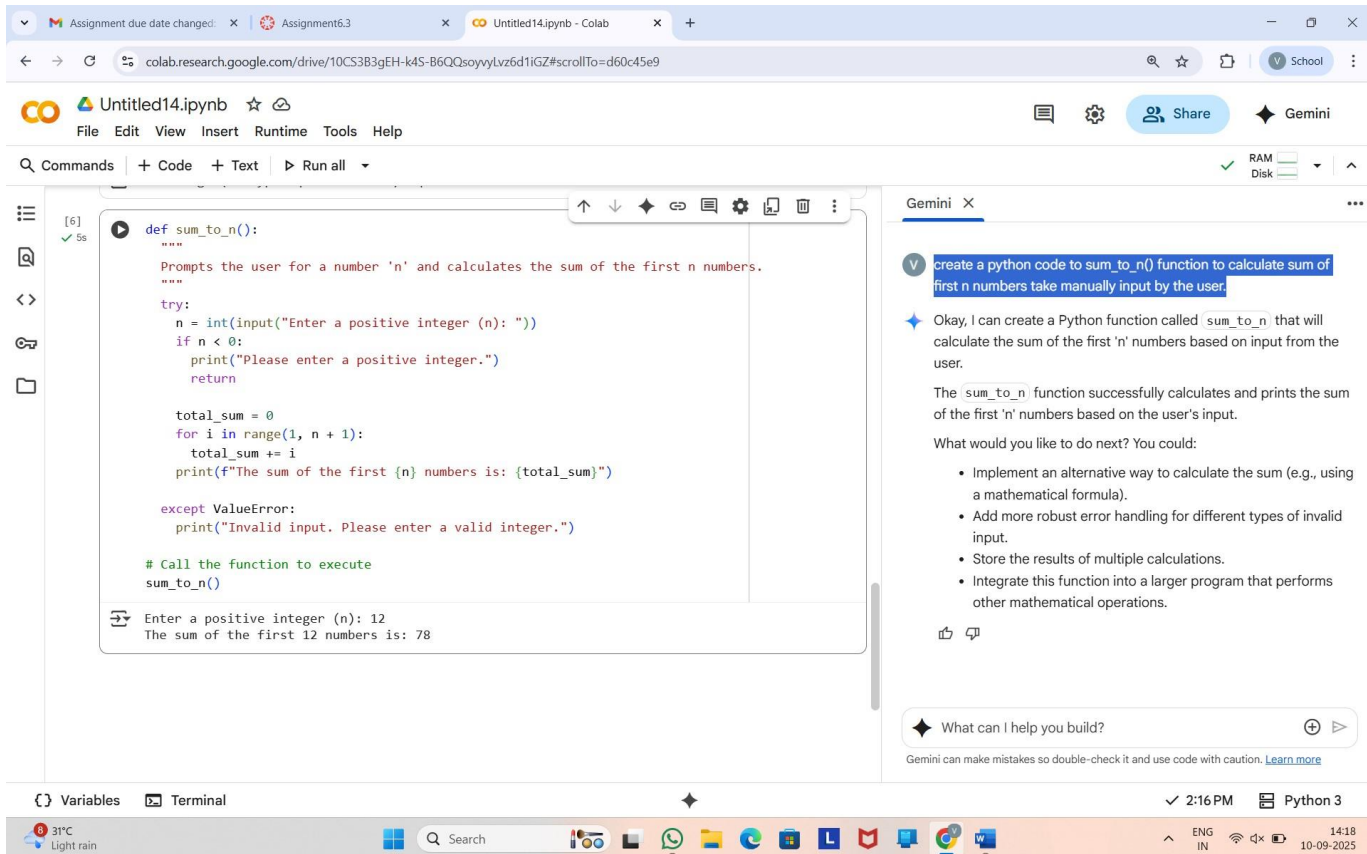
Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

**Observation:** This task successfully demonstrates how to use a while loop to repeatedly take user input and apply nested if-elif-else conditions to classify each input. It also includes a mechanism for the user to exit the input loop.

## Task-4:

**Prompt:** Create a python code to sum\_to\_n() function to calculate sum of first n numbers take manually input by the user.

## Code&Output:



The screenshot displays a Google Colab notebook titled "Untitled14.ipynb". The code defines a function `sum_to_n()` that prompts the user for a positive integer `n` and calculates the sum of the first `n` numbers. The function uses a `try` block to handle user input and a `for` loop to calculate the sum. The output shows the sum of the first 12 numbers is 78.

```
[6] ✓ 5s
def sum_to_n():
    """
    Prompts the user for a number 'n' and calculates the sum of the first n numbers.
    """
    try:
        n = int(input("Enter a positive integer (n): "))
        if n < 0:
            print("Please enter a positive integer.")
            return

        total_sum = 0
        for i in range(1, n + 1):
            total_sum += i
        print(f"The sum of the first {n} numbers is: {total_sum}")

    except ValueError:
        print("Invalid input. Please enter a valid integer.")

# Call the function to execute
sum_to_n()

Enter a positive integer (n): 12
The sum of the first 12 numbers is: 78
```

The Gemini chat window on the right shows the prompt: "create a python code to sum\_to\_n() function to calculate sum of first n numbers take manually input by the user." The response states: "Okay, I can create a Python function called 'sum\_to\_n' that will calculate the sum of the first 'n' numbers based on input from the user. The sum\_to\_n function successfully calculates and prints the sum of the first 'n' numbers based on the user's input. What would you like to do next? You could:"

- Implement an alternative way to calculate the sum (e.g., using a mathematical formula).
- Add more robust error handling for different types of invalid input.
- Store the results of multiple calculations.
- Integrate this function into a larger program that performs other mathematical operations.

What can I help you build?

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

**Observation:** This task successfully demonstrates how to define a Python function that takes user input, validates it, and uses a for loop to iterate and accumulate a sum. This is a common pattern for performing calculations over a range of numbers.

**Task-5:**

**Prompt:** Create a python code to build a BankAccount class with deposit, withdraw, and balance methods take manually input by the user.

**Code&Output:**

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyyvLz6d1IGZ#scrollTo=91555e88

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[7] 1m

```
class BankAccount:
    def __init__(self, account_number, initial_balance=0):
        self.account_number = account_number
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f'Deposit of {amount} successful. New balance: {self.balance}')
        else:
            print("Invalid deposit amount. Please enter a positive number.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f'Withdrawal of {amount} successful. New balance: {self.balance}')
        elif amount > self.balance:
            print("Insufficient funds.")
        else:
            print("Invalid withdrawal amount. Please enter a positive number.")

    def get_balance(self):
        print(f'Current balance: {self.balance}')

# Example usage with user input:
account = BankAccount(account_number="123456789")

while True:
    print("\nChoose an action:")
    print("1. Deposit")
    print("2. Withdraw")
```

Gemini

Y

Create a python code to build a BankAccount class with deposit, withdraw, and balance methods take manually input by the user.

+

Okay, I can create a Python code for a BankAccount class with deposit, withdraw, and balance methods that will take input from the user to perform these actions.

The BankAccount class with deposit, withdraw, and balance methods is working correctly, allowing the user to interact with it.

What would you like to do next? You could:

- Add more features to the BankAccount class (e.g., transfer funds, transaction history).
- Create multiple bank accounts.
- Implement a system to manage multiple bank accounts.
- Save account data to a file.
- Add more robust error handling and input validation.

What can I help you build?

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

Variables Terminal

2:22 PM Python 3

NIFTY +0.40%

Search

14:23 10-09-2025

Assignment due date changed: Assignment6.3Untitled14.ipynb - Colab

colab.research.google.com/drive/10CS3B3gEH-k4S-B6QQsoyyvLz6d1IGZ#scrollTo=91555e88

Untitled14.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[7] 1m

```
print("3. Check Balance")
print("4. Exit")

choice = input("Enter your choice (1-4): ")

if choice == '1':
    try:
        amount = float(input("Enter deposit amount: "))
        account.deposit(amount)
    except ValueError:
        print("Invalid input. Please enter a number.")
elif choice == '2':
    try:
        amount = float(input("Enter withdrawal amount: "))
        account.withdraw(amount)
    except ValueError:
        print("Invalid input. Please enter a number.")
elif choice == '3':
    account.get_balance()
elif choice == '4':
    print("Exiting.")
    break
else:
    print("Invalid choice. Please enter a number between 1 and 4.")
```

Choose an action:

1. Deposit

2. Withdraw

3. Check Balance

4. Exit

Enter your choice (1-4): 2

Enter withdrawal amount: 10000

Gemini

Y

Create a python code to build a BankAccount class with deposit, withdraw, and balance methods take manually input by the user.

+

Okay, I can create a Python code for a BankAccount class with deposit, withdraw, and balance methods that will take input from the user to perform these actions.

The BankAccount class with deposit, withdraw, and balance methods is working correctly, allowing the user to interact with it.

What would you like to do next? You could:

- Add more features to the BankAccount class (e.g., transfer funds, transaction history).
- Create multiple bank accounts.
- Implement a system to manage multiple bank accounts.
- Save account data to a file.
- Add more robust error handling and input validation.

What can I help you build?

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

Variables Terminal

2:22 PM Python 3

NIFTY +0.40%

Search

14:23 10-09-2025



The screenshot displays a Google Colab notebook titled 'Untitled14.ipynb'. The notebook contains a Python program for a simple banking application. The program prompts the user to choose an action (1. Deposit, 2. Withdraw, 3. Check Balance, 4. Exit) and processes the input. It shows a successful deposit of 120,000.0 and a withdrawal attempt of 10,000.0 that fails due to insufficient funds. The program also shows the current balance after a deposit.

```
Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): 2
Enter withdrawal amount: 10000
Insufficient funds.

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): 1
Enter deposit amount: 120000
Deposit of 120000.0 successful. New balance: 120000.0

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): 3
Current balance: 120000.0

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): 4
Exiting.
```

On the right side of the notebook, the Gemini chat interface is visible. It shows a conversation where the user asks for a Python code to build a `BankAccount` class with `deposit`, `withdraw`, and `balance` methods. Gemini responds with a code snippet and a list of suggestions for further enhancements.

**Gemini Chat:**

Y Create a python code to build a `BankAccount` class with `deposit`, `withdraw`, and `balance` methods take manually input by the user.

Okay, I can create a Python code for a `BankAccount` class with `deposit`, `withdraw`, and `balance` methods that will take input from the user to perform these actions.

The `BankAccount` class with `deposit`, `withdraw`, and `balance` methods is working correctly, allowing the user to interact with it.

What would you like to do next? You could:

- Add more features to the `BankAccount` class (e.g., transfer funds, transaction history).
- Create multiple bank accounts.
- Implement a system to manage multiple bank accounts.
- Save account data to a file.
- Add more robust error handling and input validation.

What can I help you build?

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

**Observation:** This task successfully demonstrates the creation of a Python class with multiple methods (`deposit`, `withdraw`, `get_balance`) that encapsulate data (`balance`) and behavior. It also incorporates a loop for continuous user interaction and basic error handling for invalid input, simulating a simple banking application.