

AI-Assisted Coding

Name: Vangapandla Koushik

Htno: 2403A52004

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 environment. The notebook has two tabs: 'Assignment6.3' and 'Untitled14.ipynb - Colab'. The active tab shows a Python code snippet for a `Student` class. The code defines an `__init__` method to initialize attributes, an `input_details` method to take user input, and a `display_details` method to print the attributes. An example usage is provided at the bottom of the code cell.

```
[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 AI 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 AI response states: "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:" followed by a list of suggestions:

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

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:

The screenshot shows a Google Colab notebook titled "Untitled14.ipynb". The code defines a function `print_multiples()` that prompts the user for a number and prints its first 10 multiples using a `for` loop. The function includes error handling for non-integer input. The output shows the user entered "10", and the first 10 multiples of 10 are printed: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

The Gemini AI assistant provides the following suggestions:

- Modify the function to print a different number of multiples.
- Add error handling for non-integer input.
- Store the multiples in a list instead of printing them directly.
- Explore different types of loops (e.g., `while` loop) to achieve the same result.

The bottom status bar shows the system time as 2:01 PM, the Python version as Python 3, and the weather as 31°C with light rain.

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-elif-else conditionals to classify age groups take mutiple manually input by the user.

Code&Output:

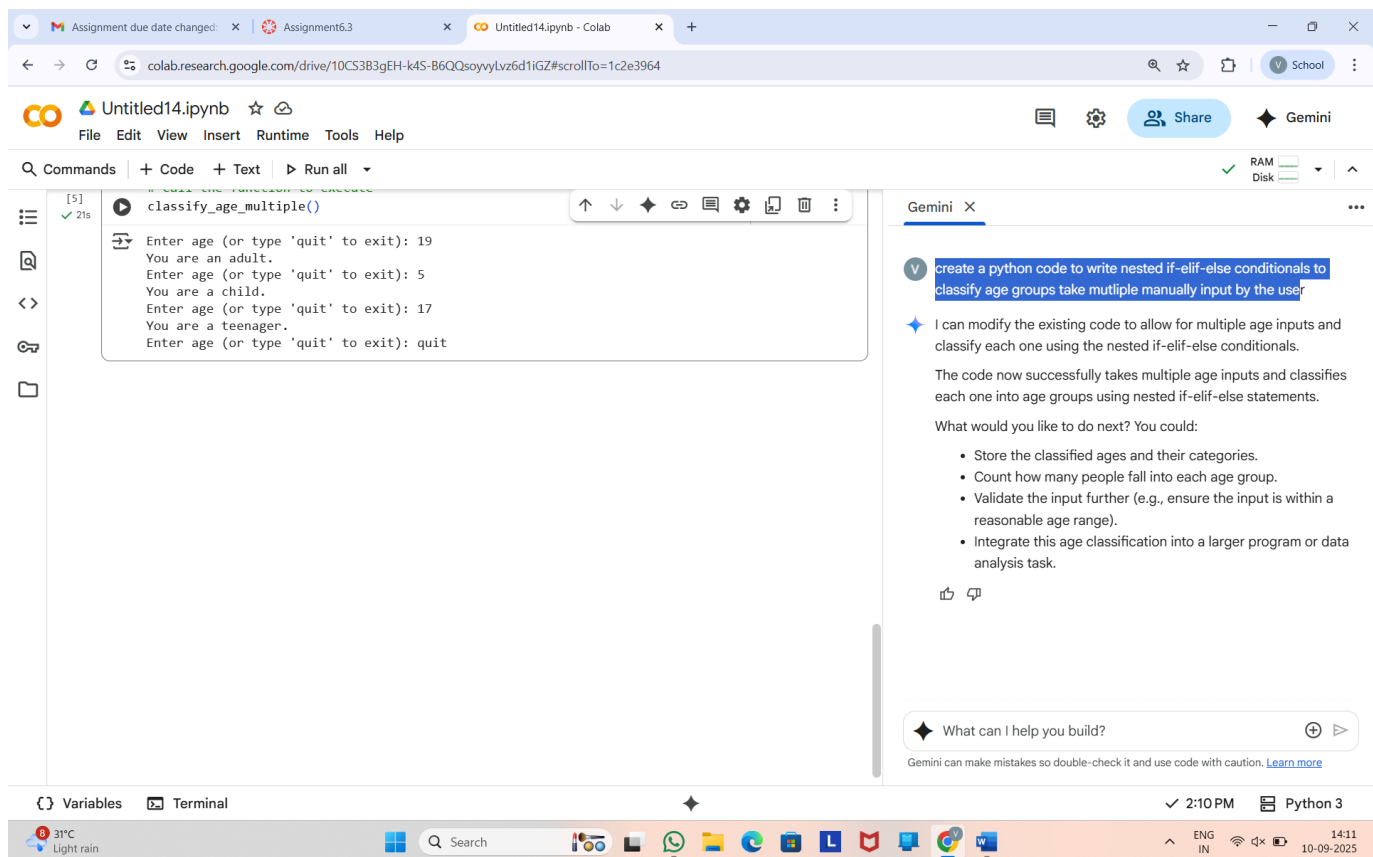
The screenshot displays a Google Colab notebook titled 'Untitled14.ipynb'. The code defines a function `classify_age_multiple()` that prompts the user for multiple ages and classifies them into age groups using nested if-elif-else conditionals. The code includes a while loop to handle multiple inputs and a try-except block to manage invalid input.

```
[5] 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()
```

The Gemini chat interface on the right shows the prompt: "create a python code to write nested if-elif-else conditionals to classify age groups take multiple manually input by the user". The response includes a confirmation that the code can be modified for multiple age inputs and a list of suggestions for further improvements:

- 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.

The bottom of the screen shows the system tray with a temperature of 31°C, light rain, and the time 2:10 PM on 10-09-2025.



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 environment. The main code cell contains a Python function `sum_to_n()` that prompts the user for a positive integer, validates it, and calculates the sum of the first `n` numbers using a `for` loop. The function is called with `n=12`, resulting in the output: "The sum of the first 12 numbers is: 78".

```
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()
```

The Gemini chat interface on the right shows a 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 confirms the creation of the `sum_to_n` function and provides suggestions for further improvements, such as using mathematical formulas or more robust error handling.

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-B6QQsoyyvLvz6d1tGZ#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

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-B6QQsoyyvLvz6d1tGZ#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

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](#)

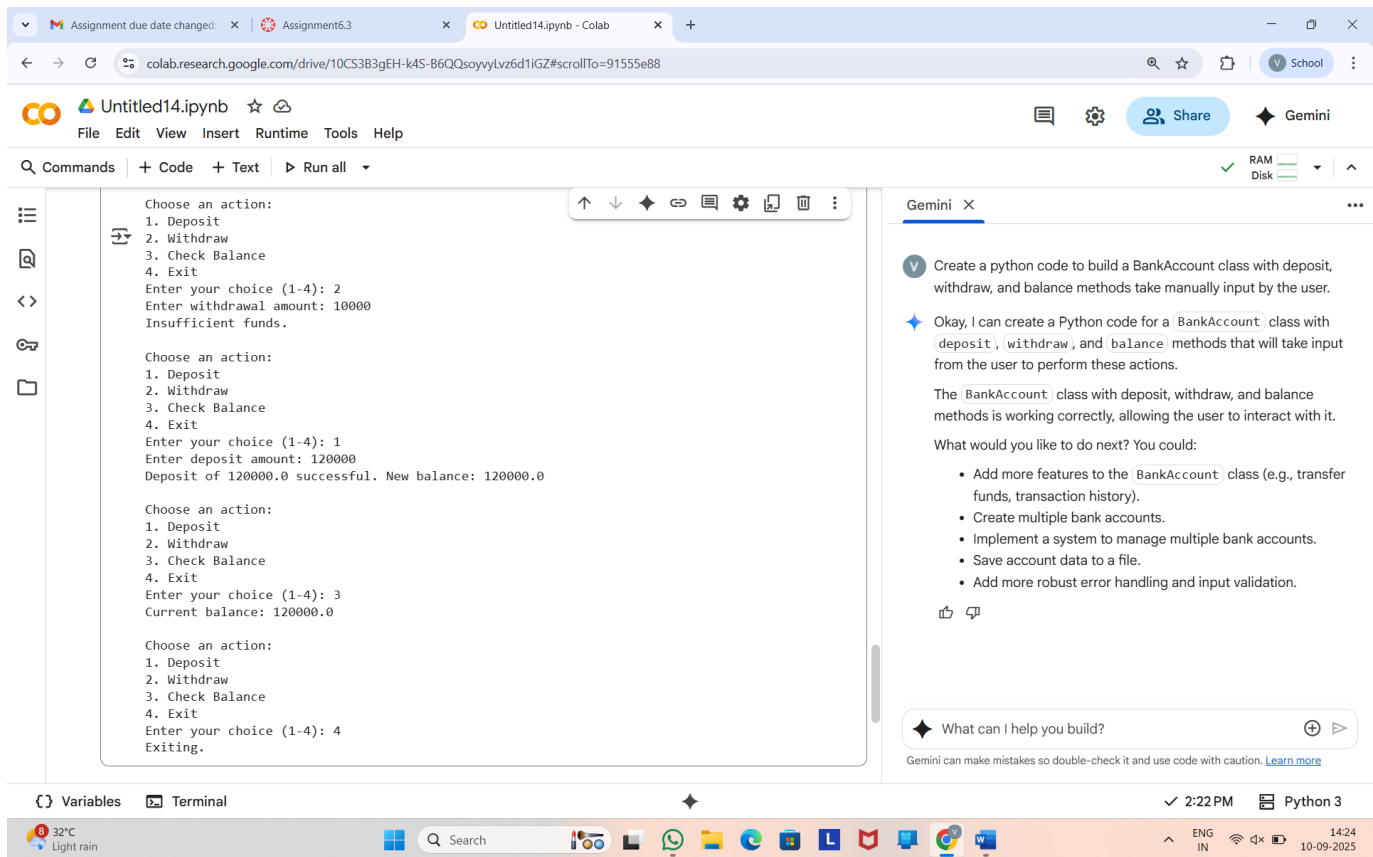
Variables Terminal

2:22 PM Python 3

NIFTY +0.40%

Search

14:23 10-09-2025



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.