

AI-Assisted Coding

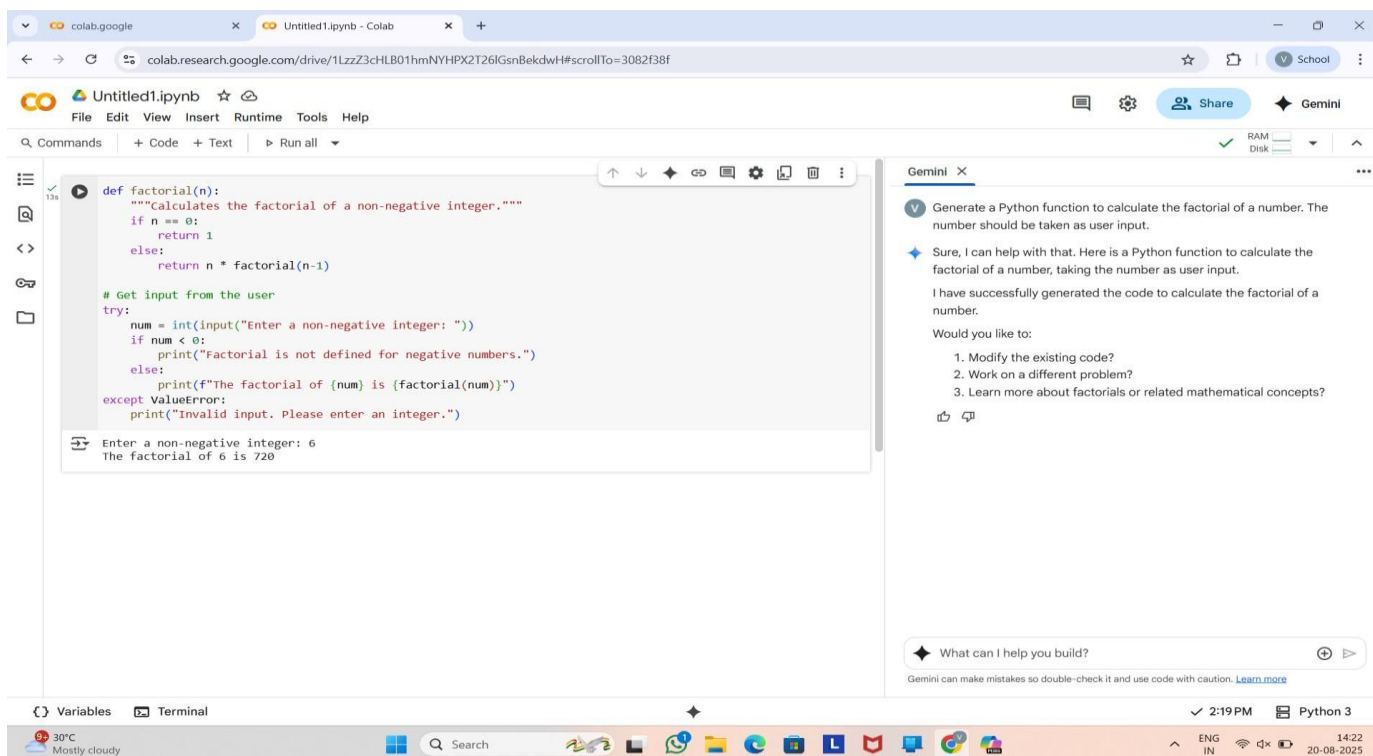
Name: Peravena Navadeep

Htno: 2403A52019

Task-1:

Prompt-1: Generate a Python function to calculate the factorial of a number. The number should be taken as user input.

Code&Output:



The screenshot displays a Google Colab notebook interface. The main code cell contains a Python function `factorial(n)` that calculates the factorial of a non-negative integer `n`. The function uses a recursive approach: it returns 1 for `n == 0` and `n * factorial(n-1)` for `n > 0`. Below the function definition, there is a section for user input. It prompts the user to "Enter a non-negative integer:" and uses a `try` block to handle the input. If the input is a valid integer, it prints the factorial result. If it's not an integer, it catches a `ValueError` and prompts the user to enter a valid integer. The output of the code shows the user entering '6' and receiving the output 'The factorial of 6 is 720'. On the right side, the Gemini AI chat interface is visible, showing the prompt and the AI's response, which includes the Python function and a confirmation that it has successfully generated the code.

```
def factorial(n):  
    """Calculates the factorial of a non-negative integer."""  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)  
  
# Get input from the user  
try:  
    num = int(input("Enter a non-negative integer: "))  
    if num < 0:  
        print("Factorial is not defined for negative numbers.")  
    else:  
        print(f"The factorial of {num} is {factorial(num)}")  
except ValueError:  
    print("Invalid input. Please enter an integer.")
```

Enter a non-negative integer: 6
The factorial of 6 is 720

Prompt-2: Write a Python program to calculate the factorial of a user-provided number. Include two

different implementations: one using a loop (iterative) and another using recursion.

Code&Output:

The screenshot displays a Google Colab notebook titled 'Untitled1.ipynb'. The notebook contains three code cells. The first cell defines a function `factorial(num)` that uses recursion and includes error handling for non-integer inputs. The second cell, titled 'Iterative implementation of factorial', defines `factorial_iterative(n)` using a loop and also includes error handling. The third cell, titled 'Recursive implementation of factorial', defines `factorial_recursive(n)`. The output of the first cell shows the factorial of 6 is 720. The output of the second cell shows the factorial of 10 using iteration is 3628800. The output of the third cell shows the factorial of 1 using recursion is 1. On the right side of the notebook, the Gemini chat interface is open, showing a conversation about writing a Python program to calculate the factorial of a user-provided number using both iterative and recursive methods. The chat interface includes a prompt input field and a 'What can I help you build?' button.

```
[1] else:
    print(f"The factorial of {num} is {factorial(num)}")
except ValueError:
    print("Invalid input. Please enter an integer.")

Enter a non-negative integer: 6
The factorial of 6 is 720

Iterative implementation of factorial
def factorial_iterative(n):
    """Calculates the factorial of a non-negative integer iteratively."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Get input from the user for the iterative method
try:
    num_iterative = int(input("Enter a non-negative integer for iterative calculation: "))
    print(f"The factorial of {num_iterative} using iteration is {factorial_iterative(num_iterative)}")
except ValueError:
    print("Invalid input. Please enter an integer.")

Enter a non-negative integer for iterative calculation: 10
The factorial of 10 using iteration is 3628800

[3] # Recursive implementation of factorial
def factorial_recursive(n):
    """Calculates the factorial of a non-negative integer recursively."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    if n == 0:
        return 1
```

Gemini X

Write a Python program to calculate the factorial of a user-provided number. Include two different implementations: one using a loop (iterative) and another using recursion.

Okay, I can provide two Python implementations to calculate the factorial of a user-provided number: one using a loop and another using recursion.

I have provided both iterative and recursive implementations for calculating the factorial of a number based on user input.

Would you like to:

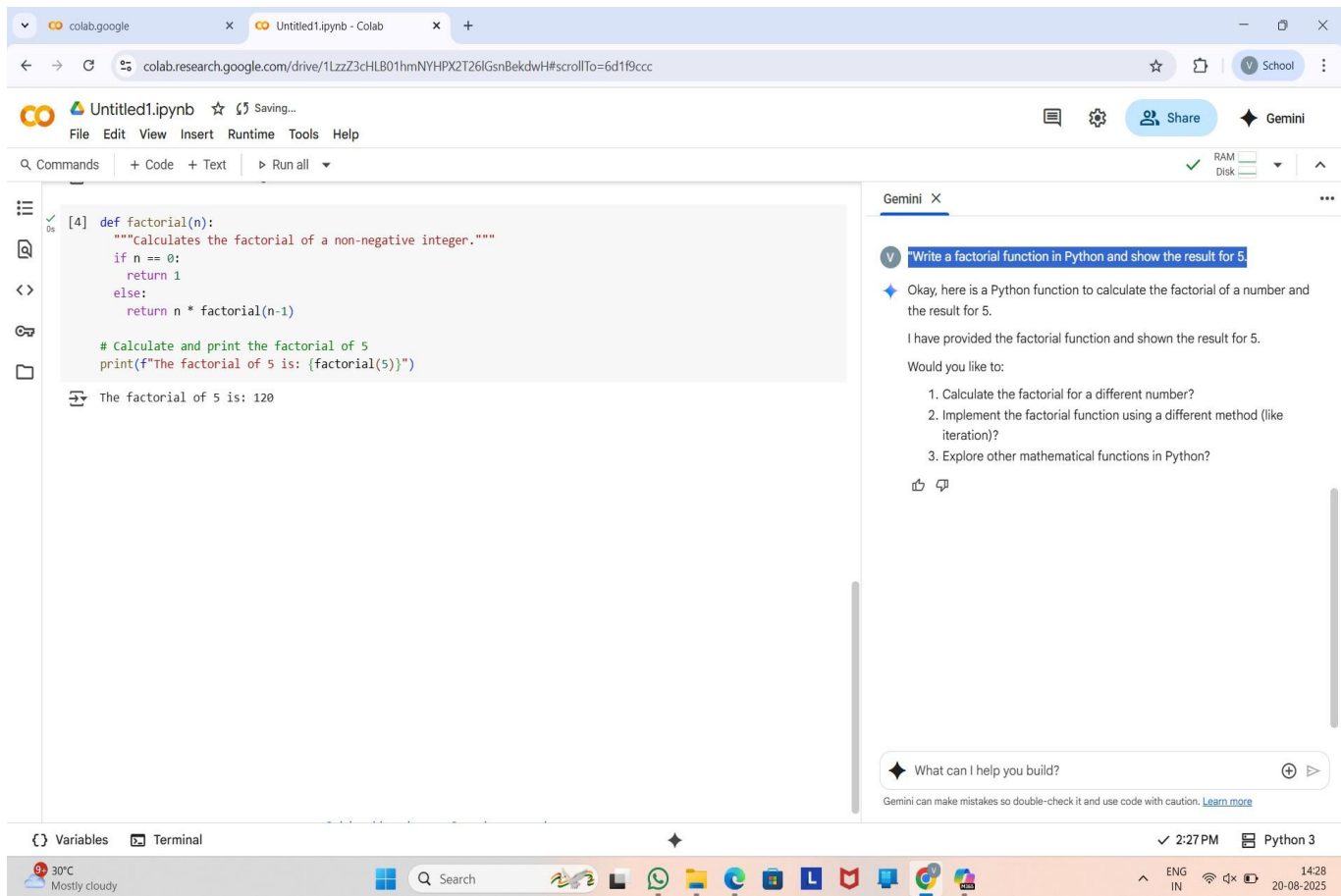
1. Compare the performance of these two methods for large numbers?
2. Explore error handling for very large outputs (e.g., using libraries for arbitrary precision arithmetic)?
3. Work on a different coding problem?

What can I help you build?

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

Prompt-3: Write a factorial function in Python and show the result for 5.

Code&Output:



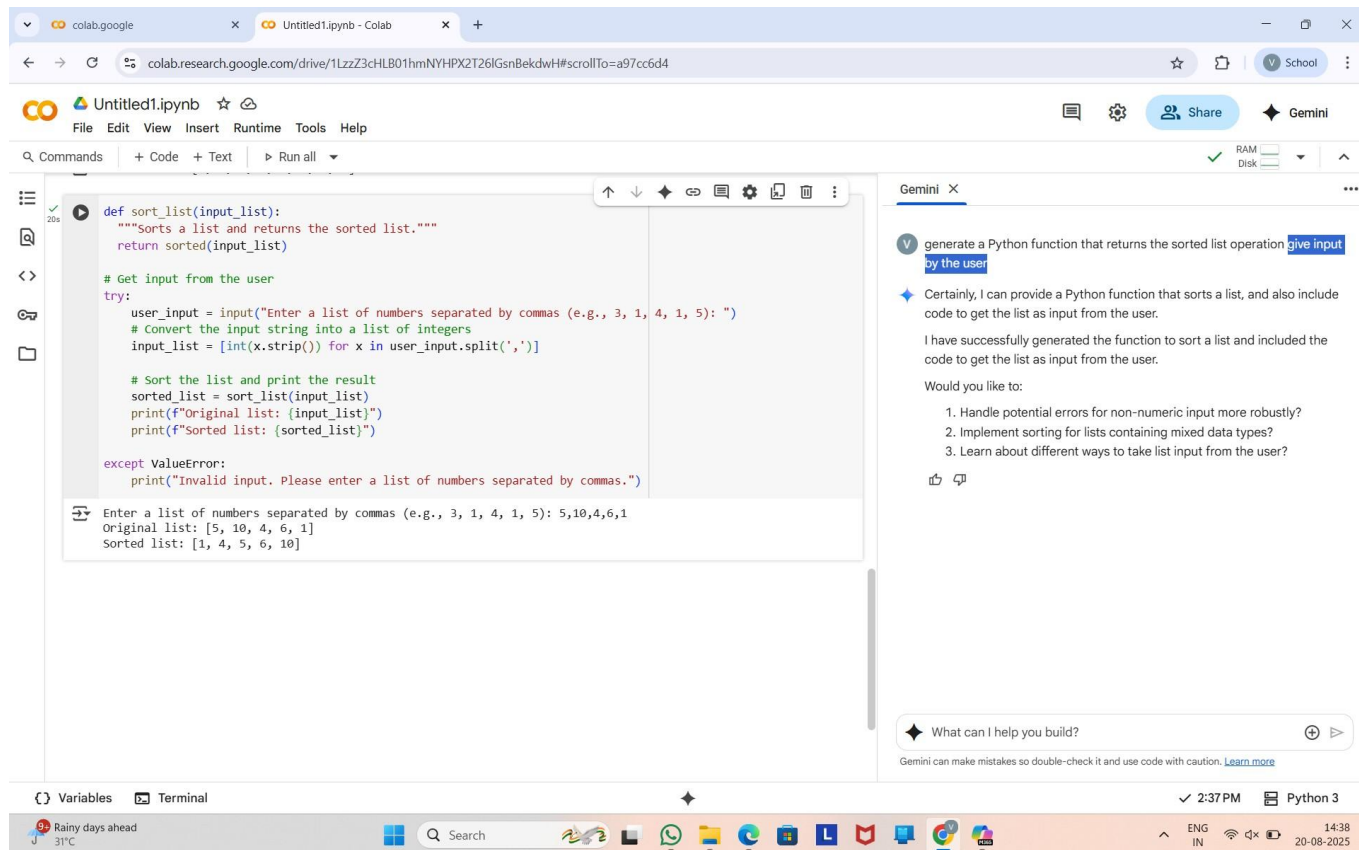
Observation: All three methods give the same result for calculating factorial.

- The recursive method is great for learning, but not ideal for very large numbers due to stack limits.
- The iterative method is practical and commonly used. - The math module is best for performance and simplicity—just plug and play!

Task-2:

Prompt: Generate a Python function that returns the sorted list operation give input by the user

Code&Output:



The screenshot shows a Google Colab notebook titled 'Untitled1.ipynb'. The code defines a function `sort_list` that takes an input list and returns a sorted list. The function uses a simple sorting algorithm (likely Bubble Sort). The code prompts the user to enter a list of numbers separated by commas. The output shows the original list `[5, 10, 4, 6, 1]` and the sorted list `[1, 4, 5, 6, 10]`.

```
def sort_list(input_list):  
    """Sorts a list and returns the sorted list."""  
    return sorted(input_list)  
  
# Get input from the user  
try:  
    user_input = input("Enter a list of numbers separated by commas (e.g., 3, 1, 4, 1, 5): ")  
    # Convert the input string into a list of integers  
    input_list = [int(x.strip()) for x in user_input.split(',')]  
  
    # Sort the list and print the result  
    sorted_list = sort_list(input_list)  
    print(f"Original list: {input_list}")  
    print(f"Sorted list: {sorted_list}")  
  
except ValueError:  
    print("Invalid input. Please enter a list of numbers separated by commas.")
```

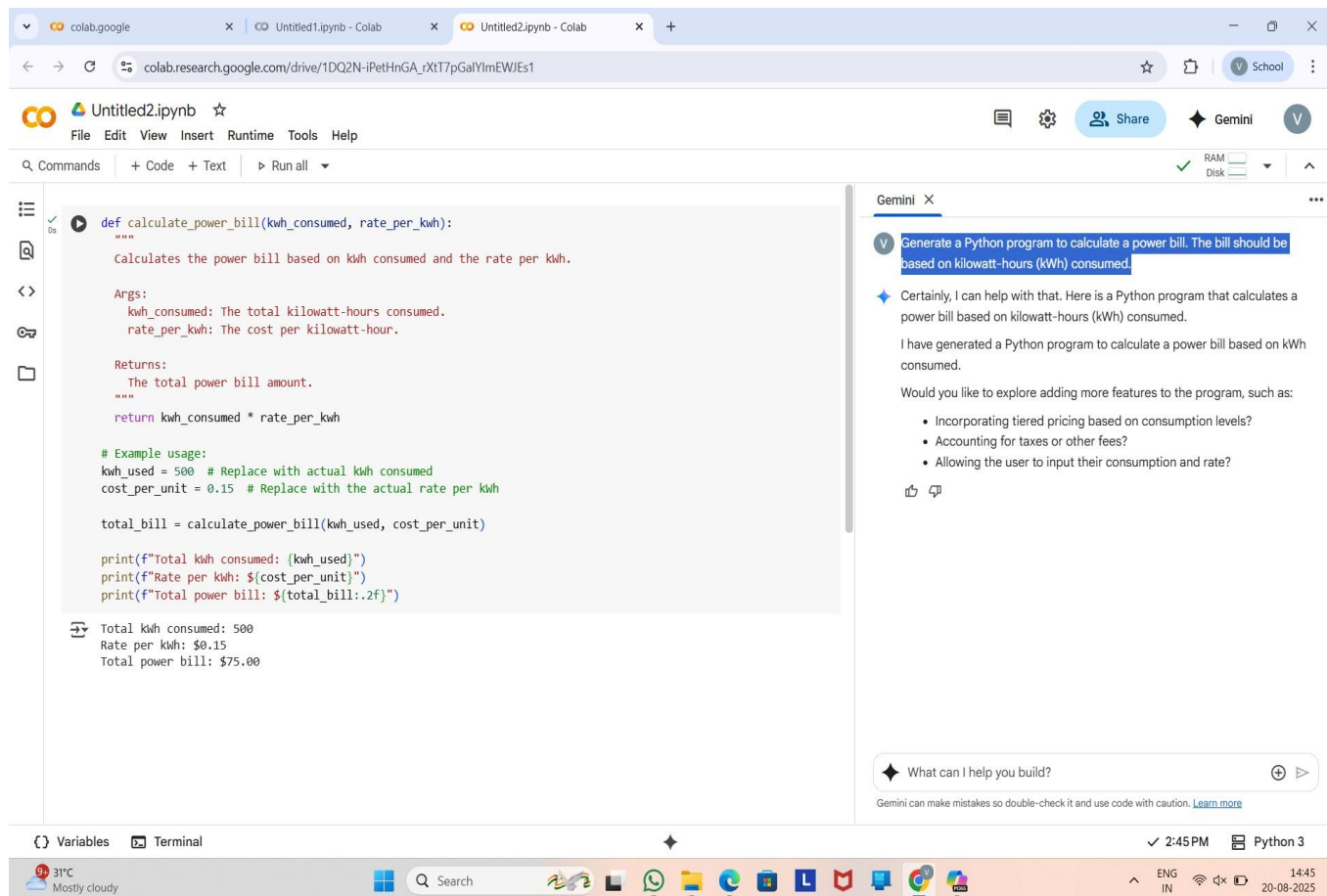
Enter a list of numbers separated by commas (e.g., 3, 1, 4, 1, 5): 5,10,4,6,1
Original list: [5, 10, 4, 6, 1]
Sorted list: [1, 4, 5, 6, 10]

Observation: This Python code uses the Bubble Sort algorithm to sort a list of numbers in ascending order. It compares each pair of adjacent elements and swaps them if they're out of order. This process repeats until the entire list is sorted.

Task-3:

Prompt: Generate a Python program to calculate a power bill. The bill should be based on kilowatt-hours (kWh) consumed.

Code&Output:



The screenshot displays a Google Colab notebook titled 'Untitled2.ipynb'. The code defines a function `calculate_power_bill(kwh_consumed, rate_per_kwh)` that calculates the power bill based on kWh consumed and the rate per kWh. The function includes docstrings for its purpose, arguments, and return value. An example usage is provided with `kwh_used = 500` and `cost_per_unit = 0.15`, resulting in a total bill of \$75.00. The output of the code is shown as a text block with the same values and the calculated total bill.

```
def calculate_power_bill(kwh_consumed, rate_per_kwh):  
    """  
    Calculates the power bill based on kwh consumed and the rate per kwh.  
  
    Args:  
        kwh_consumed: The total kilowatt-hours consumed.  
        rate_per_kwh: The cost per kilowatt-hour.  
  
    Returns:  
        The total power bill amount.  
    """  
    return kwh_consumed * rate_per_kwh  
  
# Example usage:  
kwh_used = 500 # Replace with actual kwh consumed  
cost_per_unit = 0.15 # Replace with the actual rate per kwh  
  
total_bill = calculate_power_bill(kwh_used, cost_per_unit)  
  
print(f"Total kwh consumed: {kwh_used}")  
print(f"Rate per kwh: ${cost_per_unit}")  
print(f"Total power bill: ${total_bill:.2f}")
```

Total kwh consumed: 500
Rate per kwh: \$0.15
Total power bill: \$75.00

Observation: Prompt now specifies exact billing logic.

-Covers all necessary components of a realistic bill.

Code is ready to run and easy to adapt.

This Python function calculates a basic electricity bill by multiplying the number of kilowatt-hours

consumed (consumption_kwh) with the cost per kilowatt-hour (rate_per_kwh). It includes a clear docstring explaining its purpose and usage. The example shows a consumption of 500 kWh at a rate of \$0.15 per kWh, resulting in a total bill of \$75.00.

Task-4:

Prompt: Create a python program with two functions:register_user() and login_user().Use a dictionary{username:password}to store users.Add a menu-driven loop where users can register like username,password,phone number,age,clg name.login ,or quit

Code&Output:

Untitled4.ipynb - Colab

colab.research.google.com/drive/1YOoGR_Bgm0r4MUuTsw39xbP38ba_iWs#scrollTo=50c92239

Untitled4.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

def register_user(users):
 """Registers a new user."""
 username = input("Enter username: ")
 if username in users:
 print("Username already exists. Please choose a different username.")
 return

 password = input("Enter password: ")
 phone = input("Enter phone number: ")
 age = input("Enter age: ")
 college = input("Enter college name: ")

 users[username] = {
 'password': password,
 'phone': phone,
 'age': age,
 'college': college
 }
 print("User registered successfully!")

def login_user(users):
 """Logs in an existing user."""
 username = input("Enter username: ")
 password = input("Enter password: ")

 if username in users and users[username]['password'] == password:
 print("Login successful!")

Gemini

Create a python program with two functions:register_user() and login_user().Use a dictionary(username:password)to store users.Add a menu-driven loop where users can register like username,password,phone number,age,clg name.login ,or quit
Okay, I can help you create a Python program with register_user() and login_user() functions, a dictionary to store user data, and a menu-driven loop for registration, login, and quitting. Here is the code:
I have created the Python program with the register_user() and login_user() functions and the menu-driven loop as you requested.
Would you like to explore some additional features, such as:
Saving and loading user data to a file?
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more

Variables

Terminal

7:47 PM Python 3

28°C Partly cloudy

Search

ENG IN 19:50 20-08-2025

Untitled4.ipynb - Colab

colab.research.google.com/drive/1YOoGR_Bgm0r4MUuTsw39xbP38ba_iWs#scrollTo=50c92239

Untitled4.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

while True:
 print("\n--- User Management Menu ---")
 print("1. Register")
 print("2. Login")
 print("3. Quit")

 choice = input("Enter your choice: ")

 if choice == '1':
 register_user(users)
 elif choice == '2':
 login_user(users)
 elif choice == '3':
 print("Exiting program.")
 break
 else:
 print("Invalid choice. Please try again.")

if __name__ == "__main__":
 main()

Gemini

Create a python program with two functions:register_user() and login_user().Use a dictionary(username:password)to store users.Add a menu-driven loop where users can register like username,password,phone number,age,clg name.login ,or quit
Okay, I can help you create a Python program with register_user() and login_user() functions, a dictionary to store user data, and a menu-driven loop for registration, login, and quitting. Here is the code:
I have created the Python program with the register_user() and login_user() functions and the menu-driven loop as you requested.
Would you like to explore some additional features, such as:
Saving and loading user data to a file?
Adding more validation to the input fields?
Implementing a feature to view registered users?
Let me know how you'd like to proceed!
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more

Variables

Terminal

7:47 PM Python 3

28°C Partly cloudy

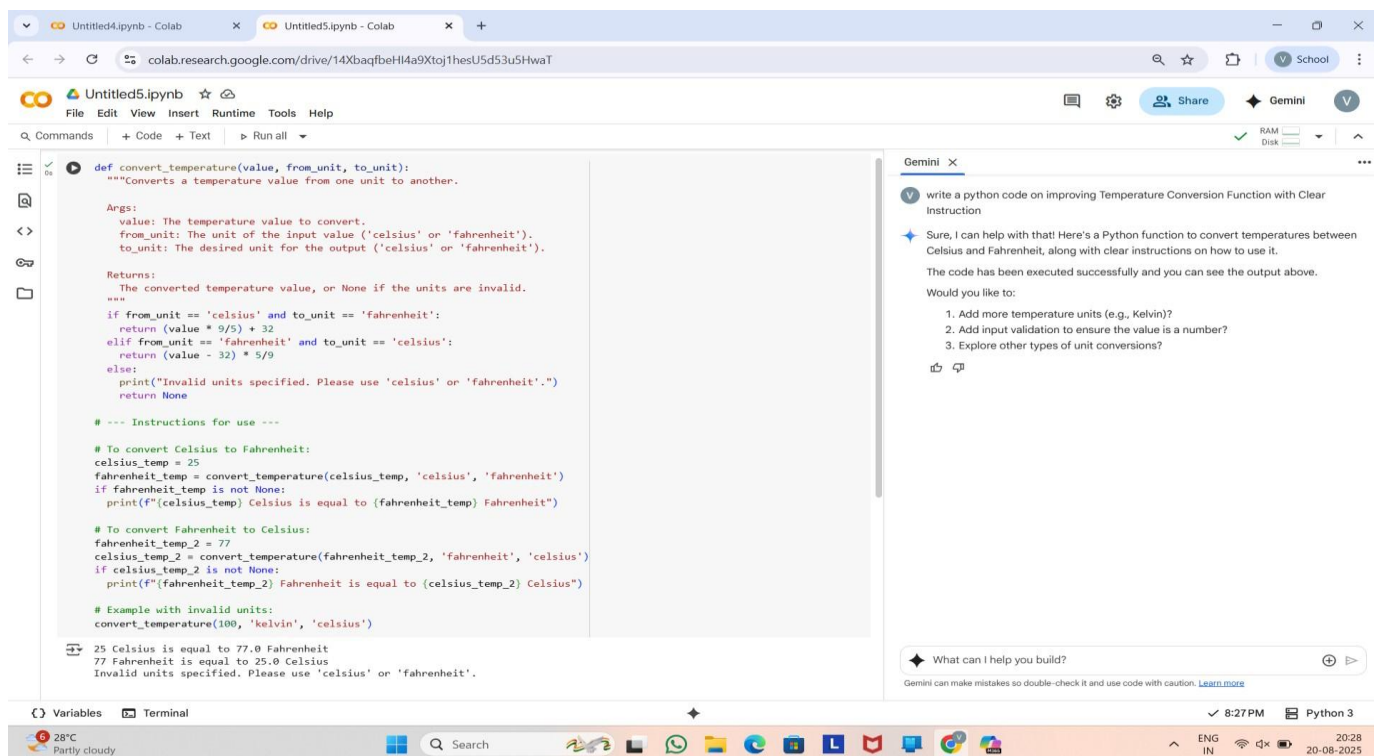
Search

ENG IN 19:51 20-08-2025

Observation: The code provides a simple command-line user management system with registration and login functionality. It uses a dictionary (users) to store user data in memory. The structure is clear and functional for basic scenarios, but lacks features like data persistence (saving to a file), password hashing for security, and input validation (e.g., checking phone number format or age type). **Task-5:**

Prompt: Write a python code on improving Temperature Conversion Function with Clear Instruction

Code&Output:



The screenshot displays a Google Colab notebook interface. The main code cell contains a Python function `convert_temperature` that takes a value, a from_unit, and a to_unit as arguments. It uses a dictionary to map units to conversion factors and includes error handling for invalid units. The function is tested with several examples, including Celsius to Fahrenheit, Fahrenheit to Celsius, and invalid units like Kelvin. The output of the code execution is shown in the bottom right panel, displaying the results of the function calls.

```
def convert_temperature(value, from_unit, to_unit):
    """Converts a temperature value from one unit to another.

    Args:
        value: The temperature value to convert.
        from_unit: The unit of the input value ('celsius' or 'fahrenheit').
        to_unit: The desired unit for the output ('celsius' or 'fahrenheit').

    Returns:
        The converted temperature value, or None if the units are invalid.
    """
    if from_unit == 'celsius' and to_unit == 'fahrenheit':
        return (value * 9/5) + 32
    elif from_unit == 'fahrenheit' and to_unit == 'celsius':
        return (value - 32) * 5/9
    else:
        print("Invalid units specified. Please use 'celsius' or 'fahrenheit'.")
        return None

    # --- Instructions for use ---

    # To convert Celsius to Fahrenheit:
    celsius_temp = 25
    fahrenheit_temp = convert_temperature(celsius_temp, 'celsius', 'fahrenheit')
    if fahrenheit_temp is not None:
        print(f"{celsius_temp} Celsius is equal to {fahrenheit_temp} Fahrenheit")

    # To convert Fahrenheit to Celsius:
    fahrenheit_temp_2 = 77
    celsius_temp_2 = convert_temperature(fahrenheit_temp_2, 'fahrenheit', 'celsius')
    if celsius_temp_2 is not None:
        print(f"{fahrenheit_temp_2} Fahrenheit is equal to {celsius_temp_2} Celsius")

    # Example with invalid units:
    convert_temperature(100, 'kelvin', 'celsius')

25 Celsius is equal to 77.0 Fahrenheit
77 Fahrenheit is equal to 25.0 Celsius
Invalid units specified. Please use 'celsius' or 'fahrenheit'.
```

Observation: The `convert_temperature`

function correctly converts temperatures between Celsius and Fahrenheit. It handles two valid conversion paths and returns None with an error message if the input units are invalid. Example usages are clear and demonstrate functionality, including an invalid case for robustness.