

**Program :**B.tech(CSE)

**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :**Bachanagari Deekshitha

**Enrollment No. :**2403A52032

**Batch No. :**02

**Date :**16/09/2025

#LAB ASSIGNMENT

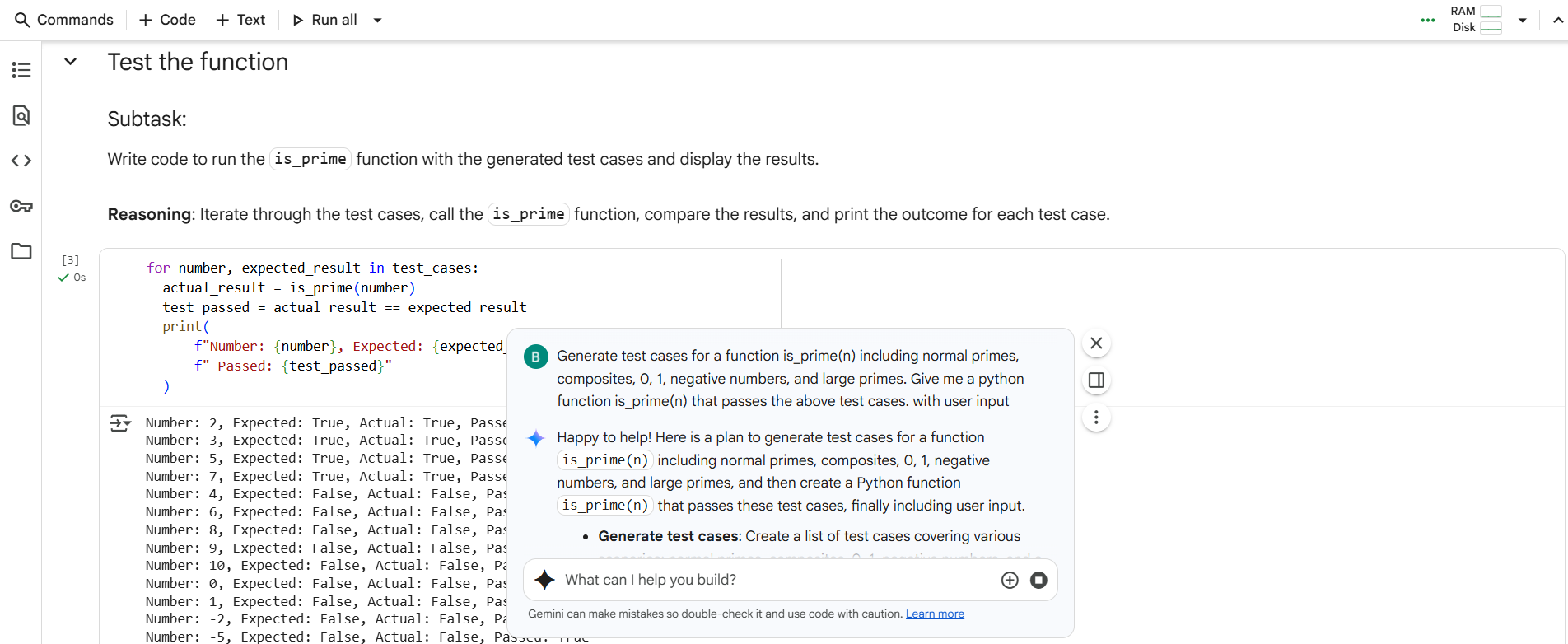
#TASK DESCRIPTION-1:

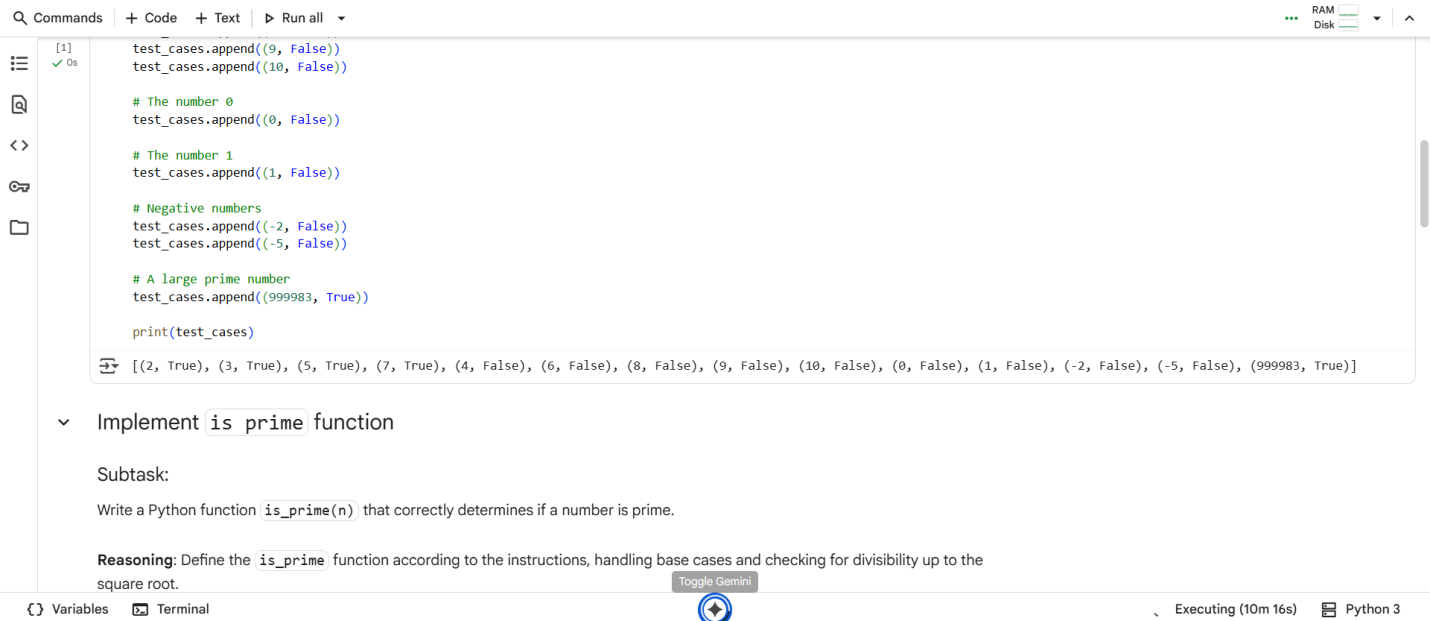
Use AI to generate test cases for a function is\_prime(n) and then implement the function.

#PROMPT:

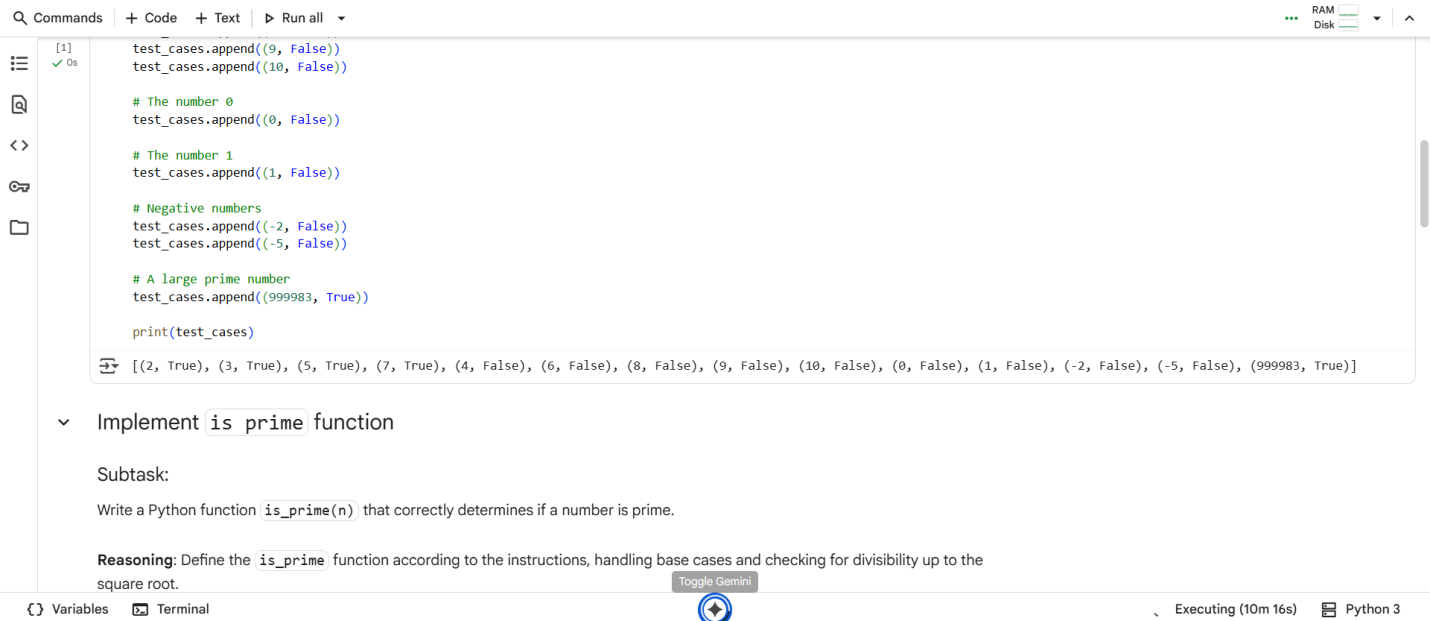
Generate test cases for a function is\_prime(n) including normal primes, composites, 0, 1, negative numbers, and large primes. Give me a python function is\_prime(n) that passes the above test cases. with user input.

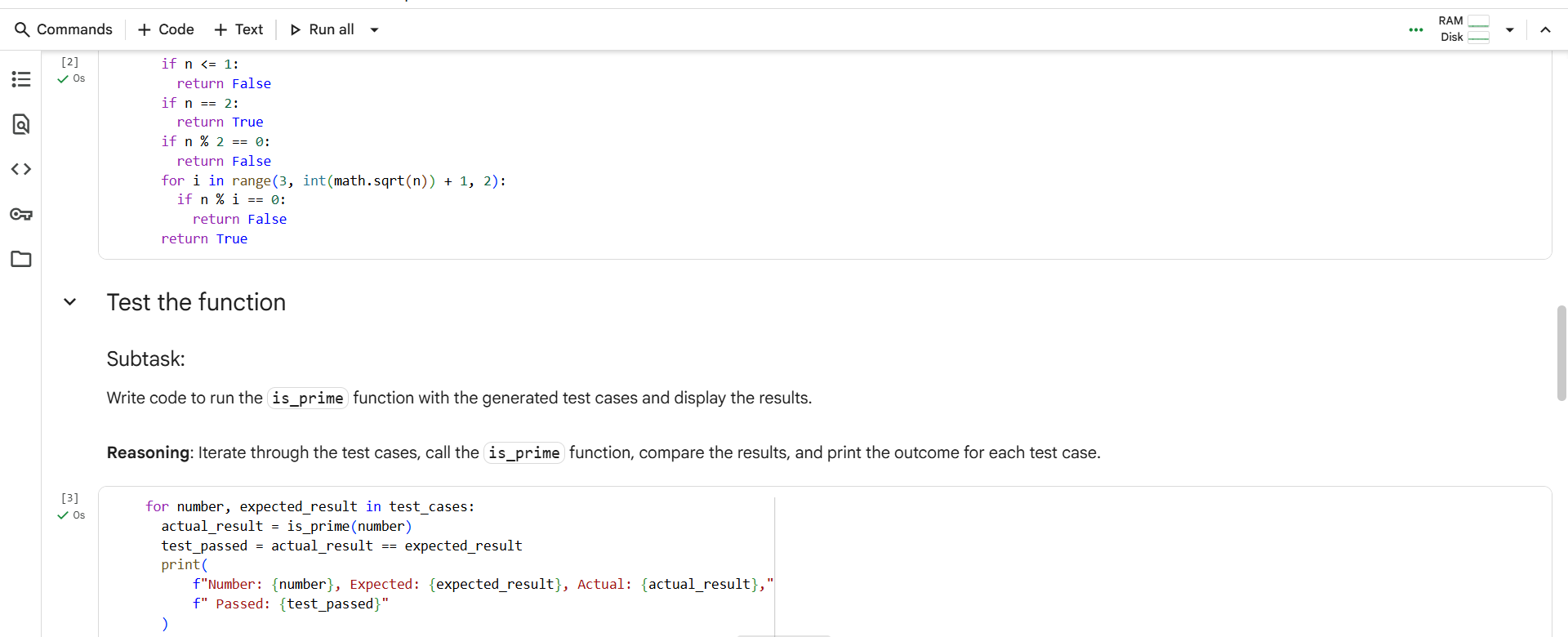
#QUESTION:

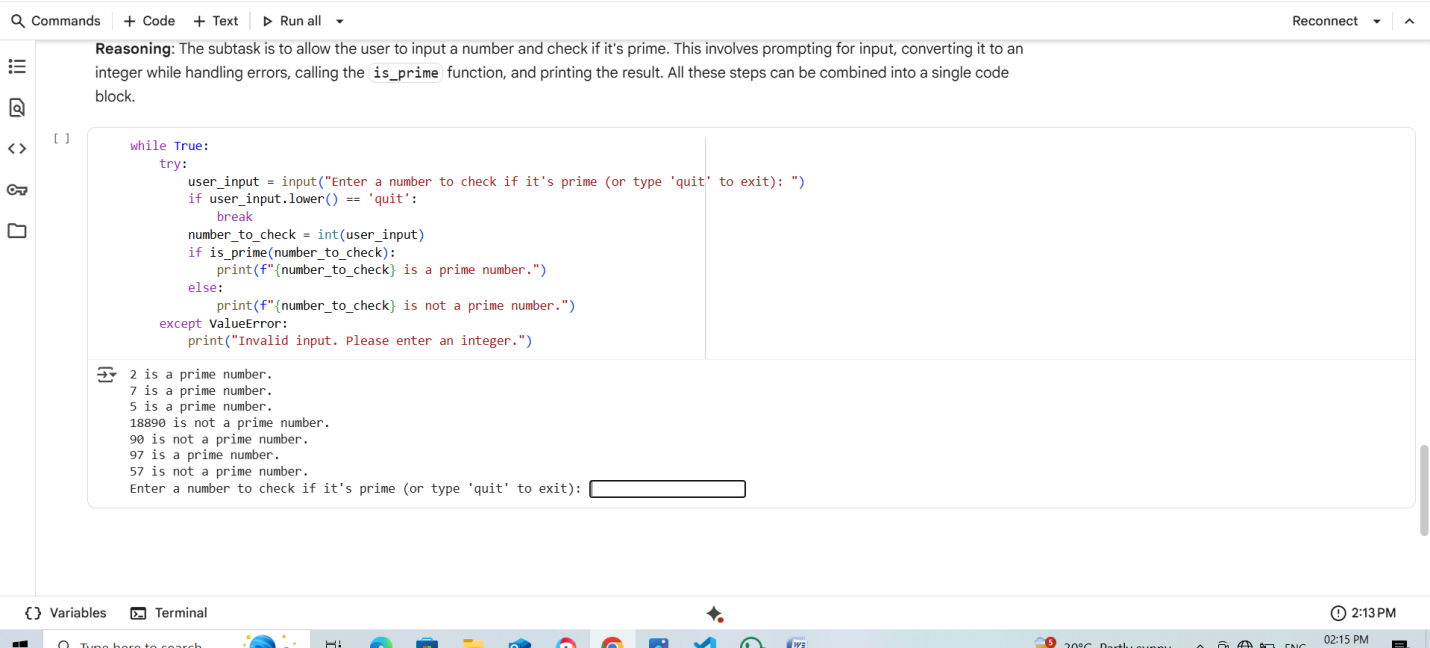




#CODE with OUTPUT:







#EXPLANATION:

This code block implements a loop that continuously prompts the user to enter a number to check if it's prime. Here's a breakdown:

* while True:: This creates an infinite loop, allowing the user to check multiple numbers without restarting the code.
* try...except ValueError:: This block handles potential errors. It tries to execute the code within the try block. If a ValueError occurs (which happens if the user enters something that cannot be converted to an integer), the code within the except block is executed.
* user\_input = input(...): This line prompts the user to enter input and stores it in the user\_input variable.
* if user\_input.lower() == 'quit': break: This checks if the user typed 'quit' (case-insensitive). If so, the break statement exits the while loop.
* number\_to\_check = int(user\_input): This line converts the user's input into an integer. If the input is not a valid integer, a ValueError is raised, and the except block is executed.
* if is\_prime(number\_to\_check): ... else: ...: This calls the is\_prime function (defined earlier) to check if the entered number is prime. Based on the function's return value (True or False), it prints a message indicating whether the number is prime or not.
* print("Invalid input. Please enter an integer."): This message is printed if a ValueError occurs, guiding the user to enter valid input.

#TASK DESCRIPTION-2:

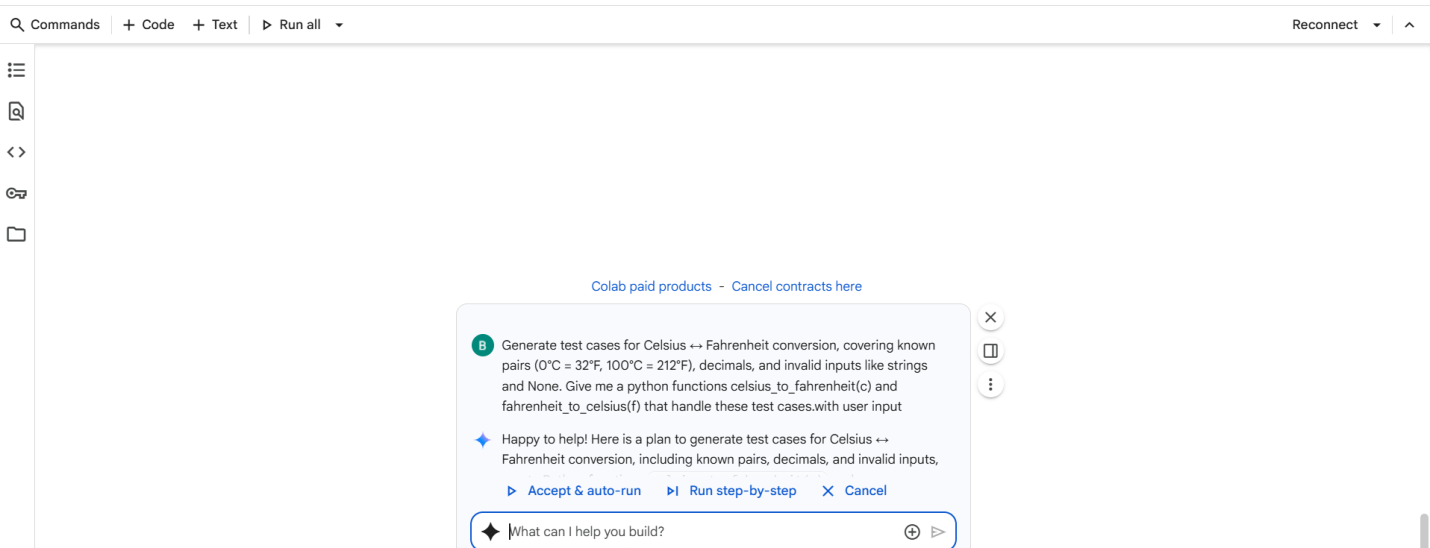
Ask AI to generate test cases for celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f).

#PROMPT:

Generate test cases for Celsius ↔ Fahrenheit conversion, covering known pairs (0°C = 32°F, 100°C = 212°F), decimals, and invalid inputs like strings and None. Give me a python functions celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f) that handle these test cases.

With user input.

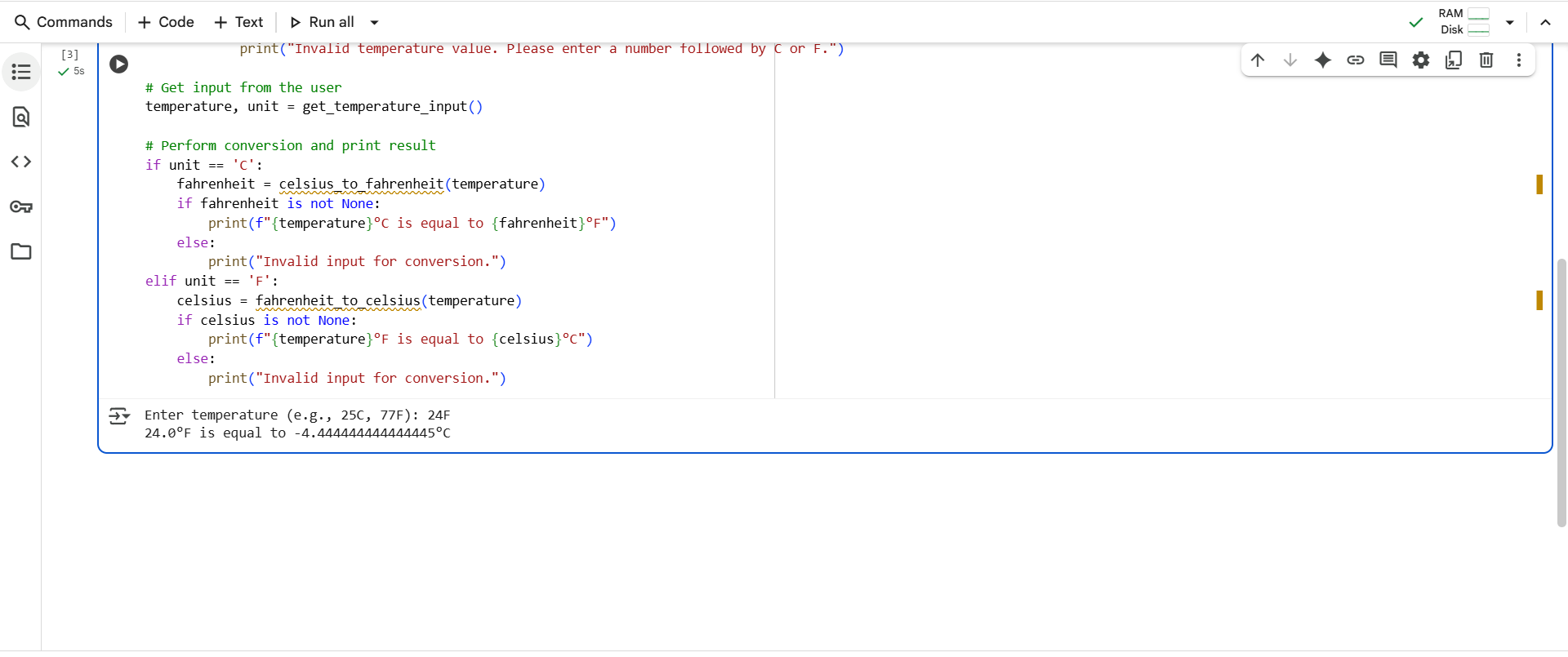
#QUESTION:



#CODE with OUTPUT:







#EXPLANATION:

This code is designed to take a temperature input from the user, determine if it's in Celsius or Fahrenheit, and then convert it to the other unit using the celsius\_to\_fahrenheit or fahrenheit\_to\_celsius functions (which were defined in a previous cell).

Here's a breakdown:

1. **def get\_temperature\_input():**: This defines a function named get\_temperature\_input. This function is responsible for getting valid temperature input from the user.
2. **while True:**: This starts an infinite loop. The loop will continue until a valid input is received and the function returns a value.
3. **temp\_str = input("Enter temperature (e.g., 25C, 77F): ")**: This line prompts the user to enter a temperature and stores their input as a string in the temp\_str variable.
4. **if not temp\_str:**: This checks if the user entered an empty string. If they did, it prints a message and the loop continues.
5. **unit = temp\_str[-1].upper()**: This extracts the last character of the input string (temp\_str[-1]) and converts it to uppercase (.upper()). This is expected to be the unit ('C' or 'F').
6. **value\_str = temp\_str[:-1]**: This extracts all characters except the last one from the input string (temp\_str[:-1]). This is expected to be the numerical value of the temperature.
7. **try...except ValueError:**: This is a try-except block to handle potential errors.
   * **value = float(value\_str)**: Inside the try block, it attempts to convert the extracted value\_str into a floating-point number.
   * **if unit == 'C': return value, 'C'**: If the unit is 'C', it returns the numerical value and the unit 'C'.
   * **elif unit == 'F': return value, 'F'**: If the unit is 'F', it returns the numerical value and the unit 'F'.
   * **else: print("Invalid unit...")**: If the unit is not 'C' or 'F', it prints an error message.
   * **except ValueError: print("Invalid temperature value...")**: If the float(value\_str) conversion fails (e.g., the user entered non-numeric characters before the unit), a ValueError occurs, and this block prints an error message.
8. **temperature, unit = get\_temperature\_input()**: This line calls the get\_temperature\_input() function and unpacks the returned tuple into the temperature and unit variables.
9. **if unit == 'C': ... elif unit == 'F': ...**: This checks the unit variable.
   * If the unit is 'C', it calls the celsius\_to\_fahrenheit() function to convert the temperature.
   * If the unit is 'F', it calls the fahrenheit\_to\_celsius() function to convert the temperature.
10. **if fahrenheit is not None: ... else: ...** and **if celsius is not None: ... else: ...**: These checks ensure that the conversion functions returned a valid number (not None, which would indicate invalid input to the conversion function). If the conversion was successful, it prints the result in a user-friendly format. Otherwise, it indicates invalid input for conversion.

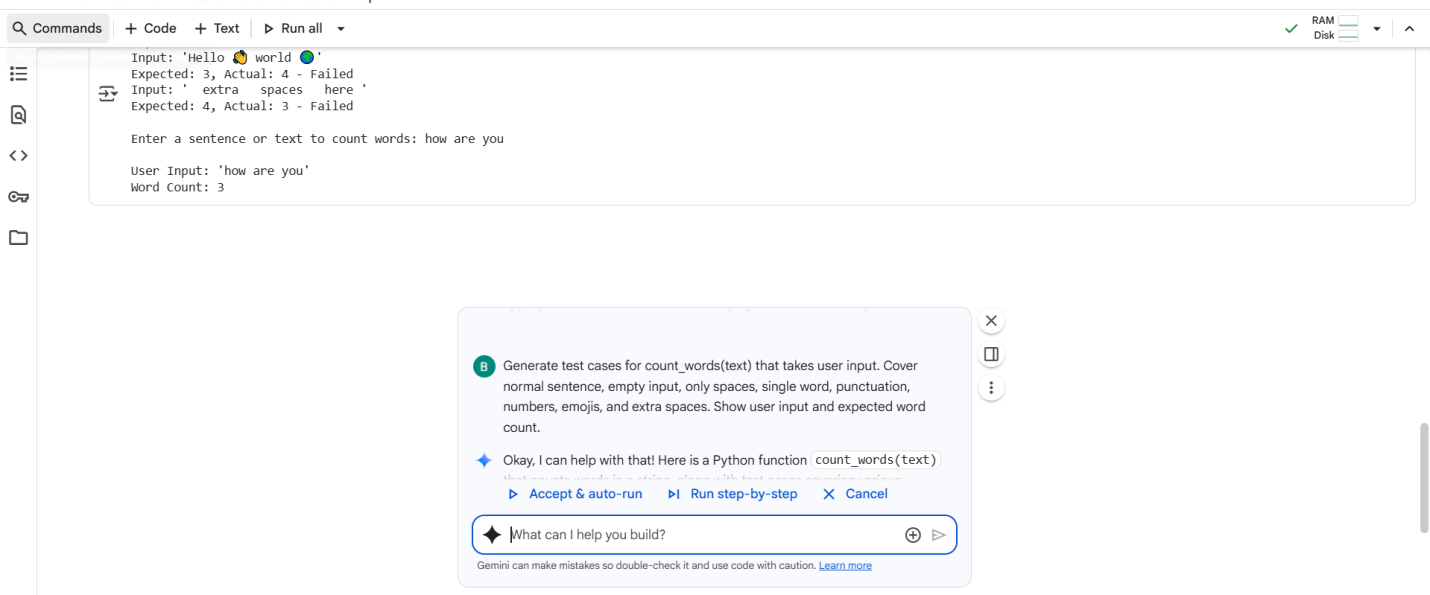
#TASK DESCRIPTION-3:

Use AI to write test cases for a function count\_words(text) that returns the number of words in a sentence.

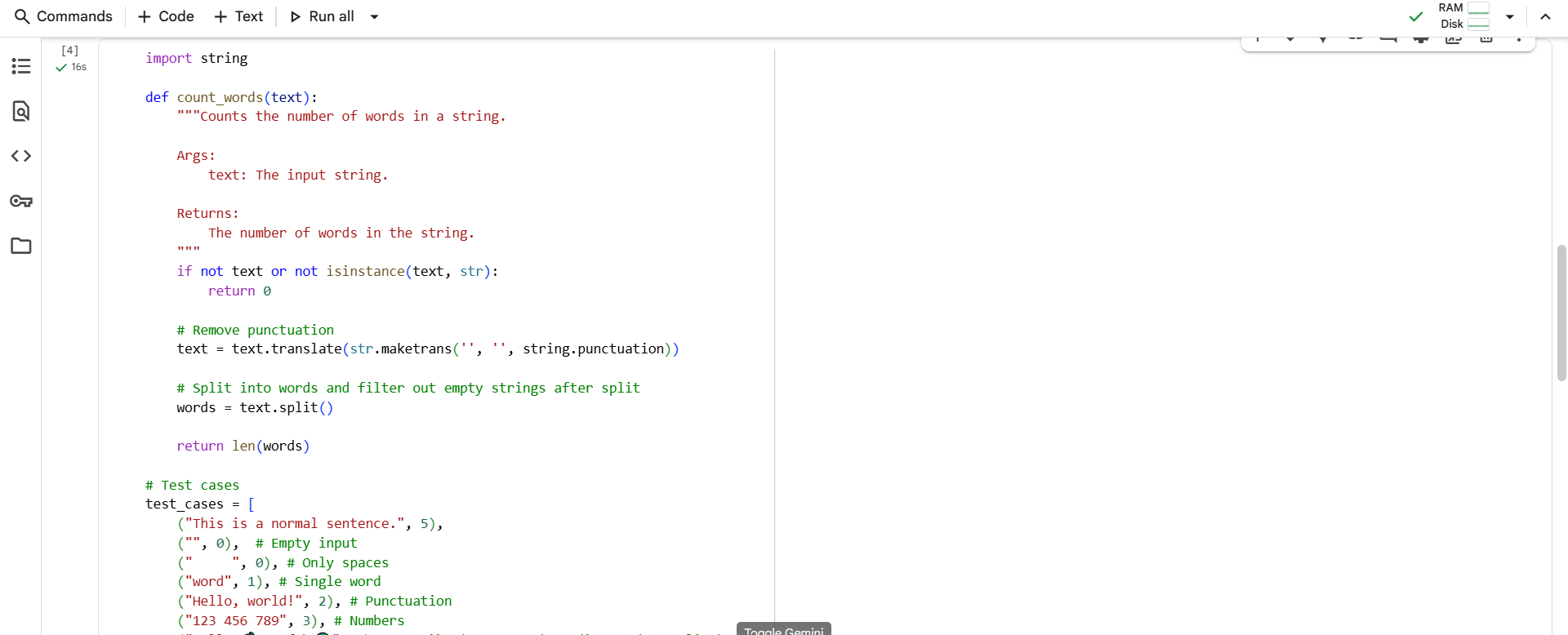
#PROMPT:

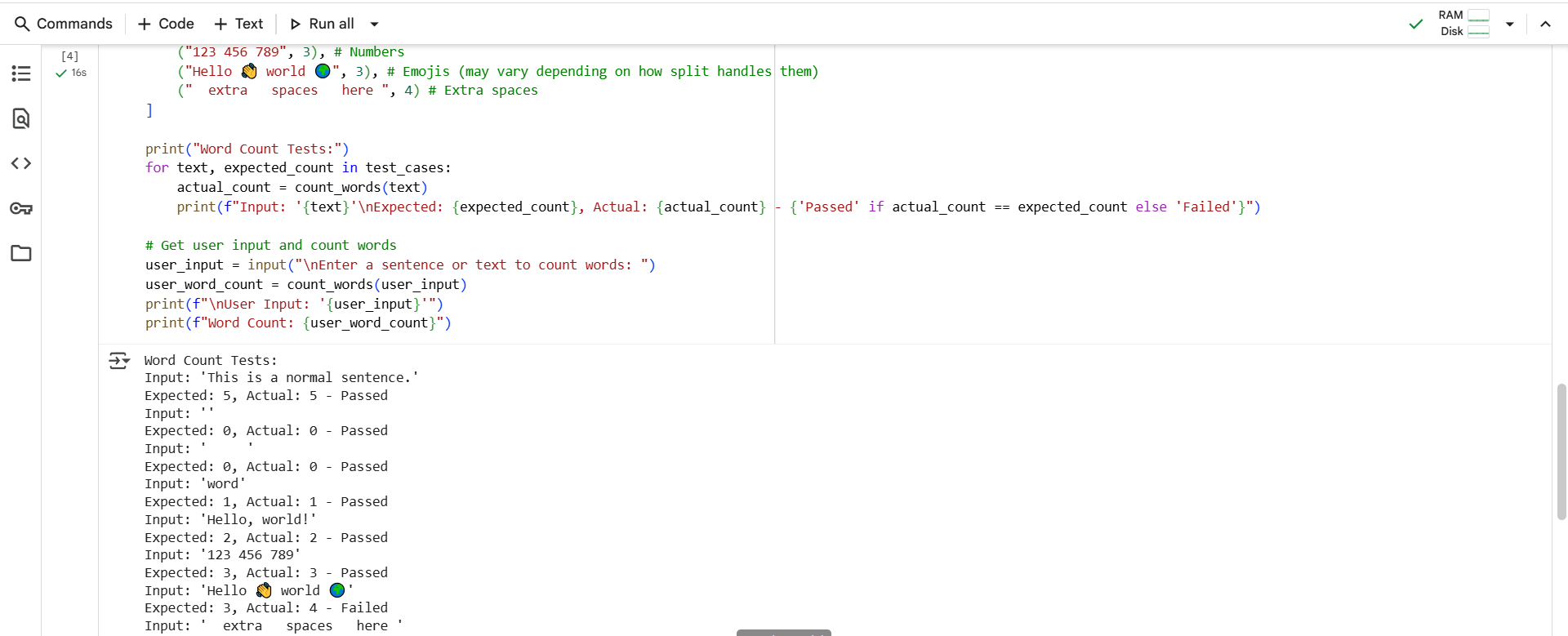
Generate test cases for count\_words(text) that takes user input. Cover normal sentence, empty input, only spaces, single word, punctuation, numbers, emojis, and extra spaces. Show user input and expected word count.

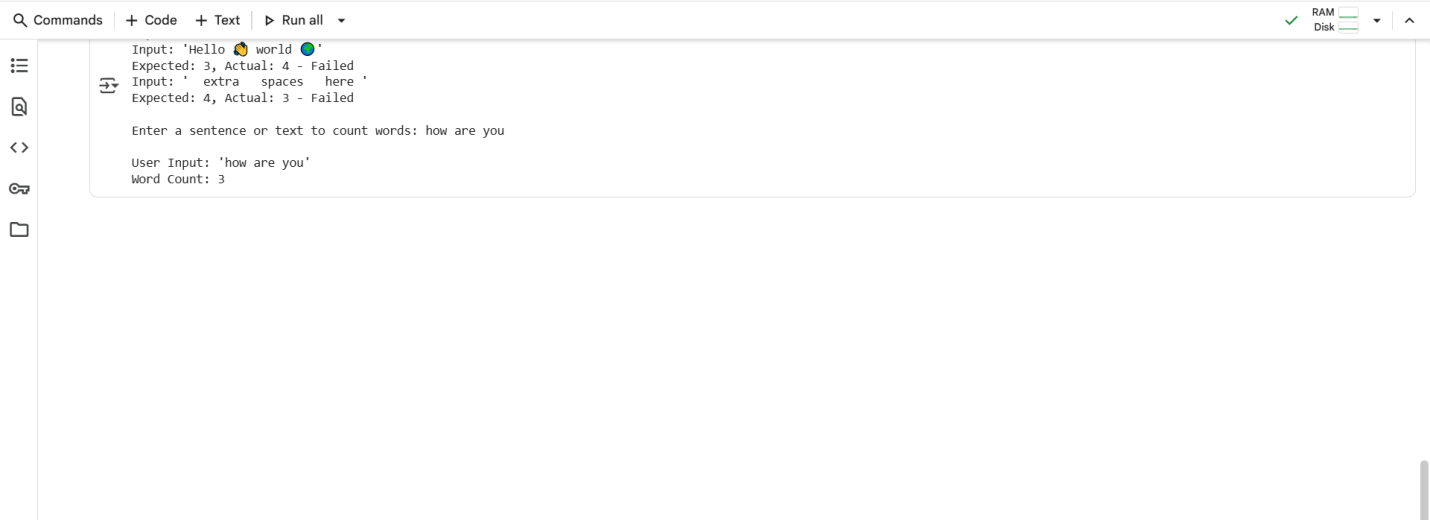
#QUESTION:



#CODE with OUTPUT:







#TASK DESCRIPTION-4:

Generate test cases for a BankAccount class with:

**Methods:**

deposit(amount)

withdraw(amount)

check\_balance()

**Requirements:**

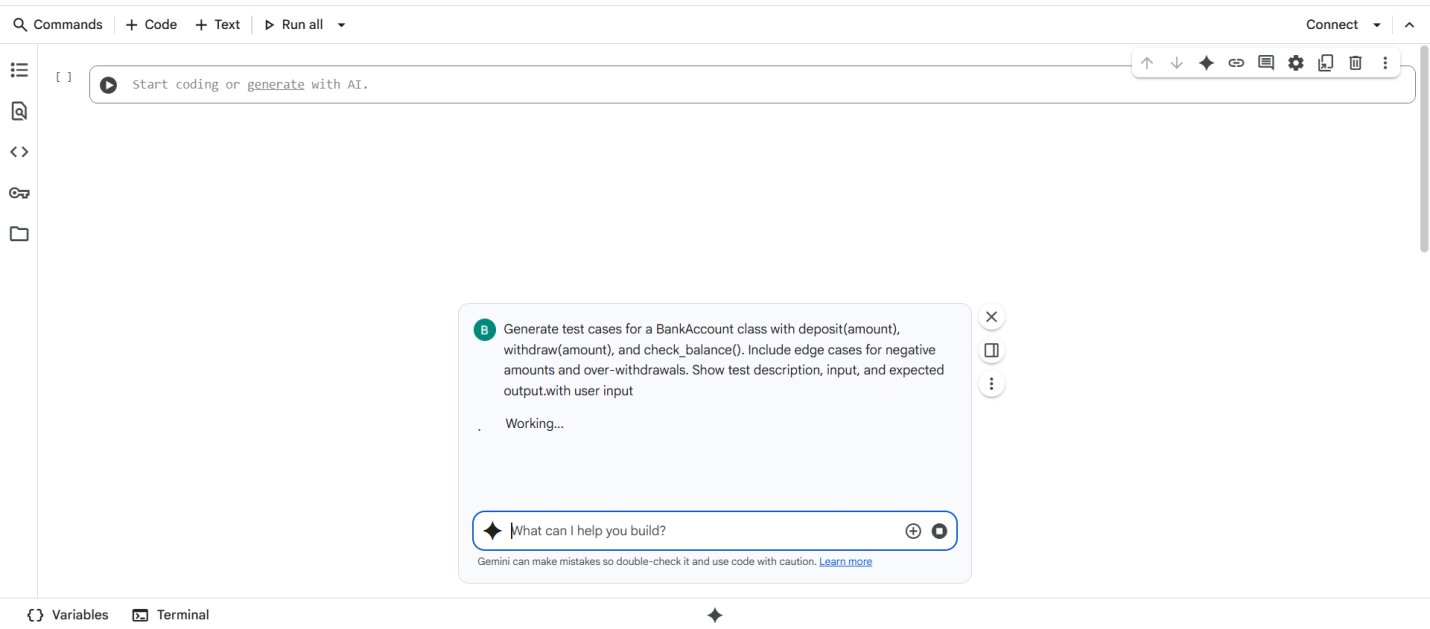
Negative deposits/withdrawals should raise an error.

Cannot withdraw more than balance.

#PROMPT:

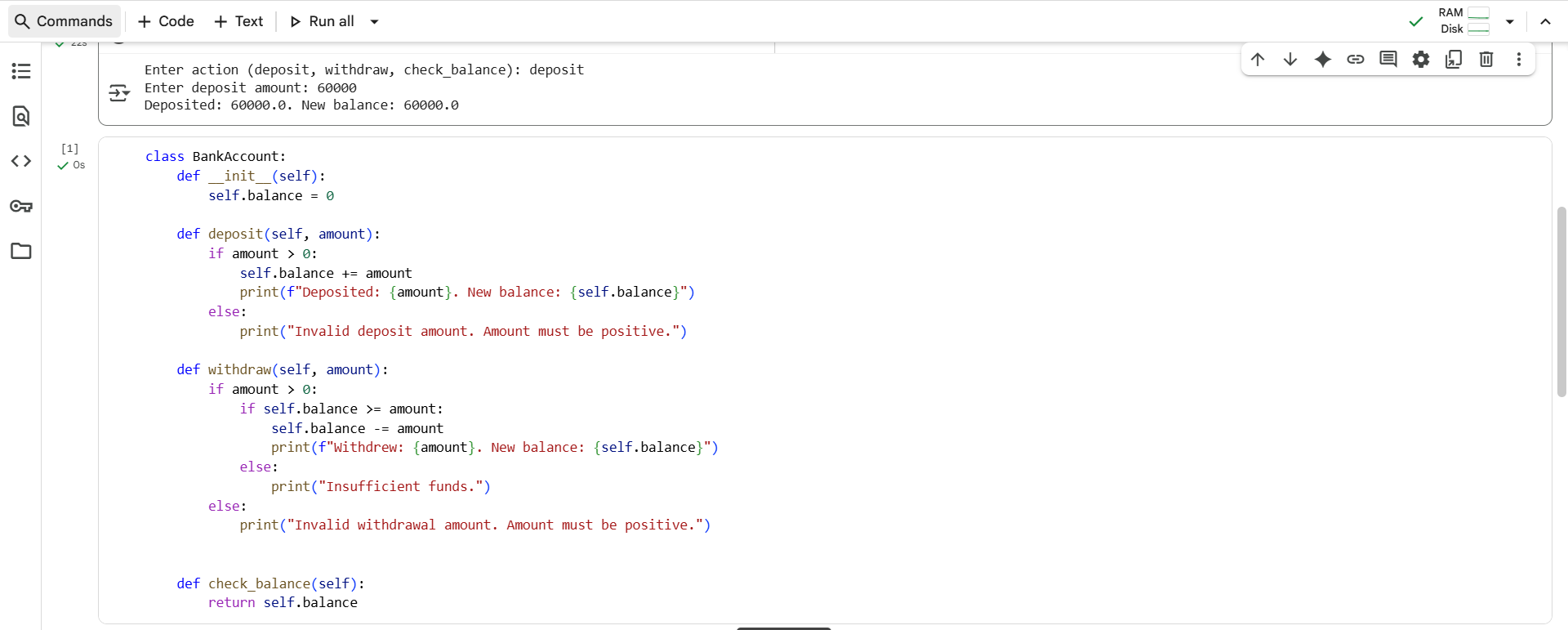
Generate test cases for a BankAccount class with deposit(amount), withdraw(amount), and check\_balance(). Include edge cases for negative amounts and over-withdrawals. Show test description, input, and expected output.

#QUESTION:



#CODE with OUTPUT:





#EXPLANATION:

* **class BankAccount:**: This line declares the beginning of a new class named BankAccount. Classes are blueprints for creating objects (in this case, bank accounts) that have specific properties (like a balance) and behaviors (like depositing or withdrawing).
* **def \_\_init\_\_(self):**: This is the constructor method. It's called automatically when you create a new BankAccount object. The self parameter refers to the instance of the class being created. Inside the constructor, self.balance = 0 initializes the bank account's balance to zero when a new account is created.
* **def deposit(self, amount):**: This method handles depositing money into the account.
  + if amount > 0:: It checks if the deposit amount is positive.
  + self.balance += amount: If the amount is positive, it adds the amount to the current self.balance.
  + print(f"Deposited: {amount}. New balance: {self.balance}"): It prints a confirmation message showing the deposited amount and the new balance.
  + else: print("Invalid deposit amount. Amount must be positive."): If the amount is not positive, it prints an error message.
* **def withdraw(self, amount):**: This method handles withdrawing money from the account.
  + if amount > 0:: It checks if the withdrawal amount is positive.
  + if self.balance >= amount:: If the amount is positive, it then checks if there are sufficient funds in the account (self.balance is greater than or equal to the amount to be withdrawn).
  + self.balance -= amount: If there are sufficient funds, it subtracts the amount from the current self.balance.
  + print(f"Withdrew: {amount}. New balance: {self.balance}"): It prints a confirmation message showing the withdrawn amount and the new balance.
  + else: print("Insufficient funds."): If there are insufficient funds, it prints an "Insufficient funds" message.
  + else: print("Invalid withdrawal amount. Amount must be positive."): If the withdrawal amount is not positive, it prints an error message.
* **def check\_balance(self):**: This method simply returns the current balance of the account (self.balance).

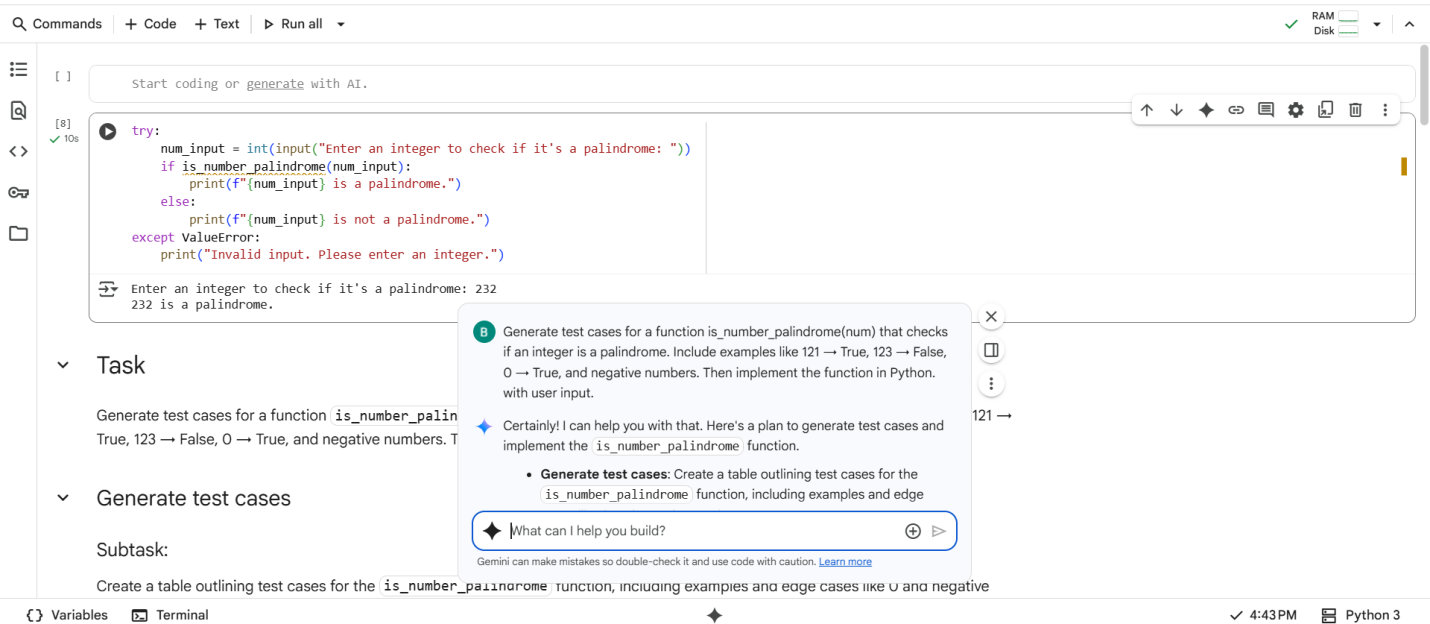
#TASK DESCRIPTION-5:

Generate test cases for is\_number\_palindrome(num), which checks if an integer reads the same backward.

#PROMPT:

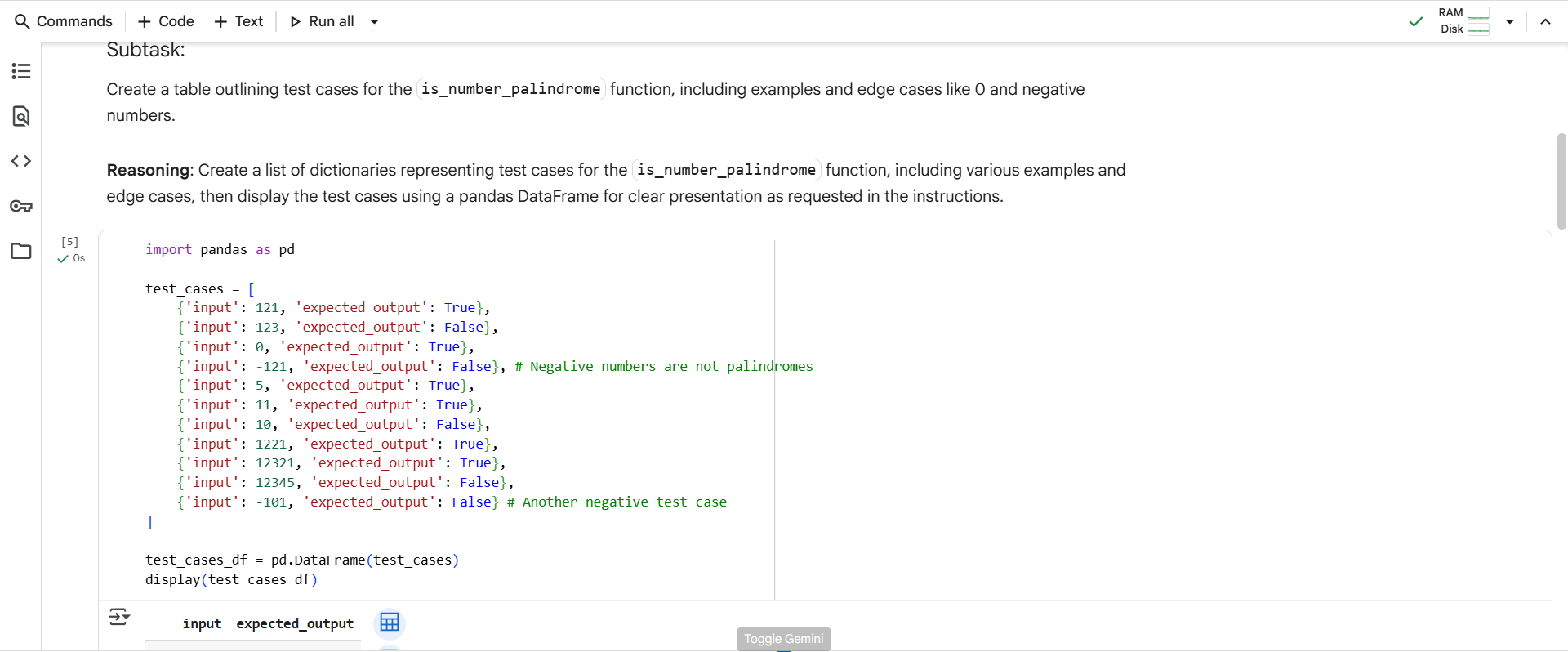
Generate test cases for a function is\_number\_palindrome(num) that checks if an integer is a palindrome. Include examples like 121 → True, 123 → False, 0 → True, and negative numbers. Then implement the function in Python. with user input.

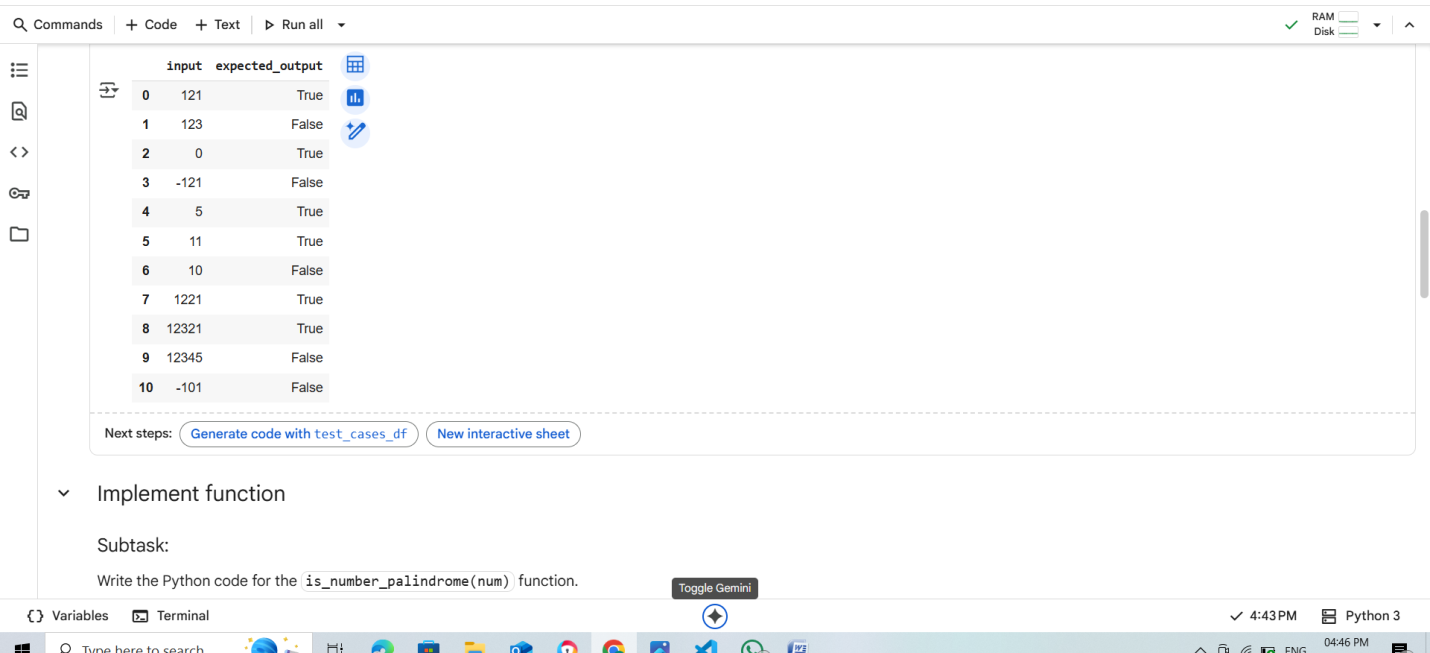
#QUESTION:

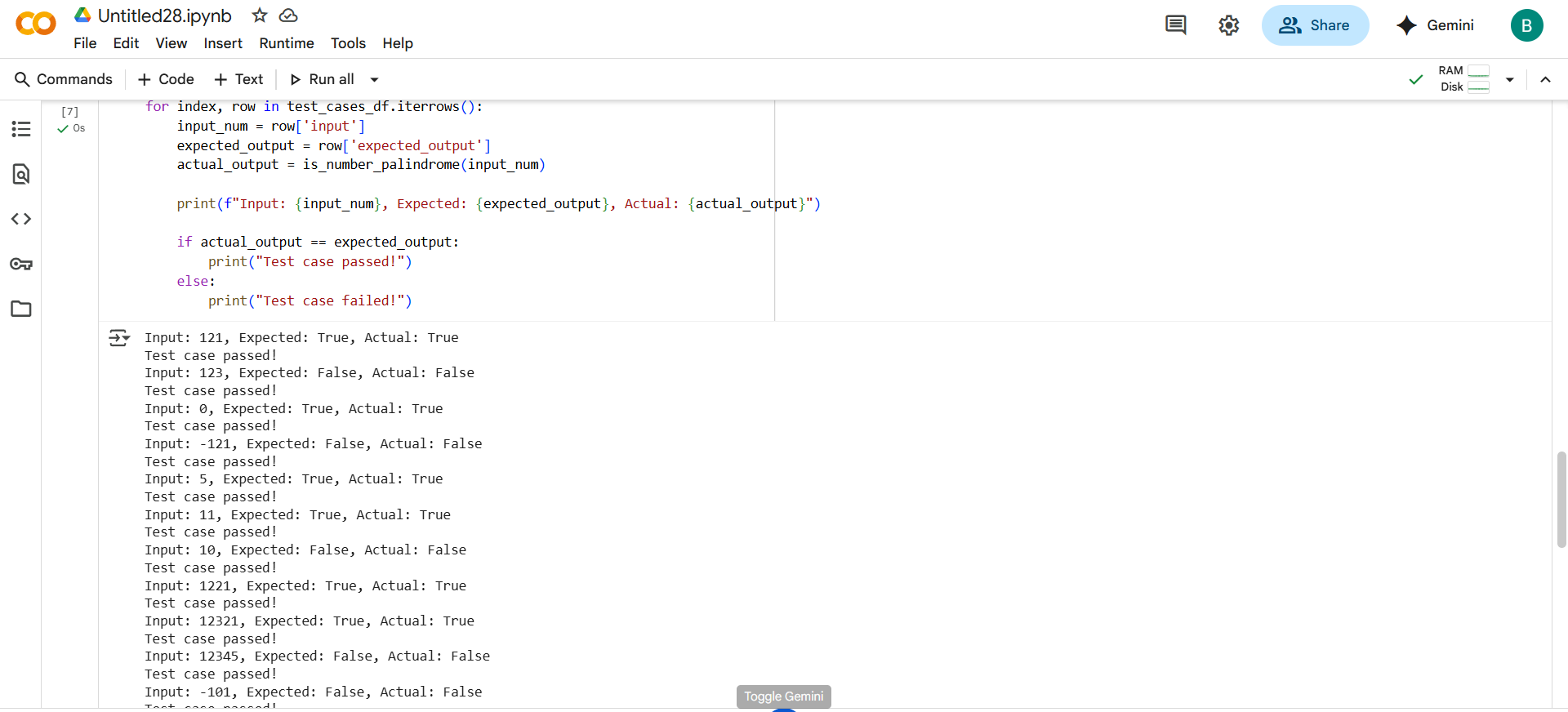


#CODE with OUTPUT:









#EXPLANATION:

* **for index, row in test\_cases\_df.iterrows():**: This loop iterates through each row of the test\_cases\_df DataFrame. In each iteration:
  + index gets the index of the current row.
  + row gets a Series object representing the data in the current row.
* **input\_num = row['input']**: This line extracts the value from the 'input' column of the current row and assigns it to the variable input\_num. This is the number that will be passed to the is\_number\_palindrome function.
* **expected\_output = row['expected\_output']**: This line extracts the value from the 'expected\_output' column of the current row and assigns it to the variable expected\_output. This is the expected result for the given input, according to our test cases.
* **actual\_output = is\_number\_palindrome(input\_num)**: This line calls the is\_number\_palindrome function with the input\_num and stores the returned value (either True or False) in the actual\_output variable.
* **print(f"Input: {input\_num}, Expected: {expected\_output}, Actual: {actual\_output}")**: This line prints the input number, the expected output, and the actual output from the function call for the current test case. This makes it easy to see the results of each test.
* **if actual\_output == expected\_output:**: This conditional statement compares the actual\_output from the function with the expected\_output from the test case.
  + **print("Test case passed!")**: If the actual output matches the expected output, it prints "Test case passed!".
  + **else: print("Test case failed!")**: If the actual output does not match the expected output, it prints "Test case failed!".

THANK YOU