PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTED CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : KARNAKANTI GODADEVI

ENROLLMENT NO : 2403A52003

BATCH NO : 01

Task Description#1 (Classes)

- Use AI to complete a Student class with attributes and a method.
- Check output
- Analyze the code generated by AI tool.

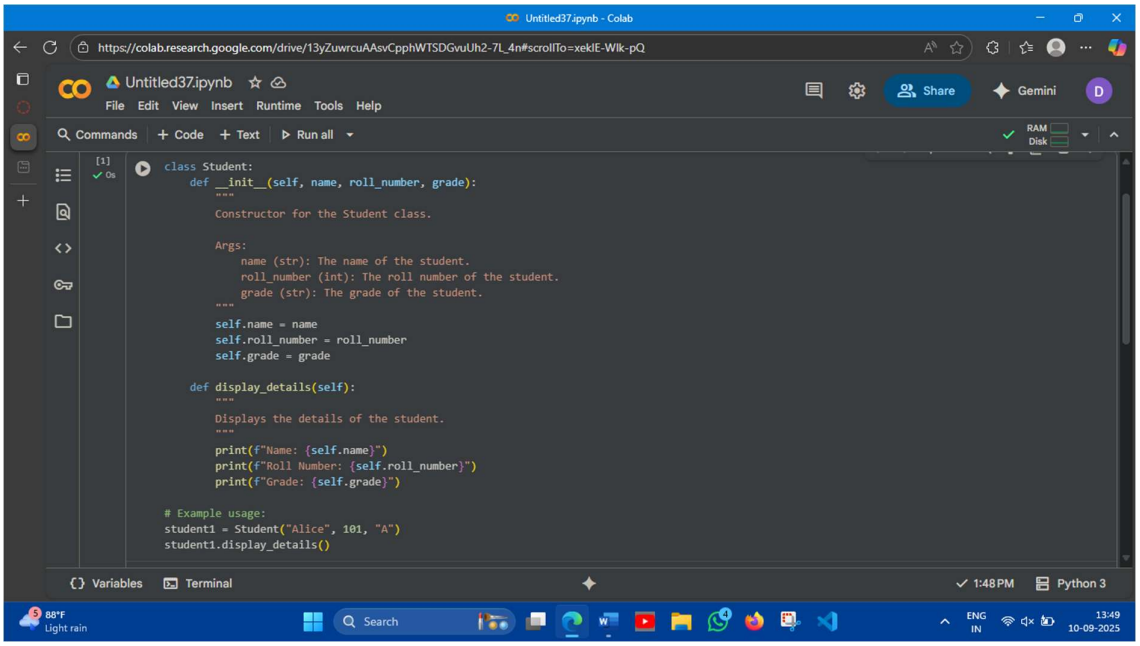
Expected Output#1

- Class with constructor and display_details() method

Prompt:

generate a code for student class with constructor , attributes and display_details() method

Code&output:



Name: Alice

Roll Number: 101

Grade: A

Code Analysis:

- **class Student:: This line defines a new class named Student. Classes are blueprints for creating objects (instances).**
- **def __init__(self, name, roll_number, grade):: This is the constructor method. It's called automatically when you create a new Student object.**
    - **self: Refers to the instance of the class being created.**
    - **name, roll_number, grade: These are parameters that you pass when creating a Student object.**
    - **Inside the constructor, self.name = name, self.roll_number = roll_number, and self.grade = grade assign the values passed as arguments to the object's attributes (properties).**
- **def display_details(self):: This defines a method named display_details within the Student class. Methods are functions that belong to a class.**

- **self: Again, refers to the instance of the class.**
- **This method prints the name, roll_number, and grade attributes of the Student object in a formatted string.**

- **student1 = Student("Alice", 101, "A"): This line creates an instance of the Student class.**
  - **Student("Alice", 101, "A") calls the constructor with the values "Alice", 101, and "A" for the name, roll_number, and grade parameters, respectively.**
  - **The resulting Student object is assigned to the variable student1.**
- **student1.display_details(): This line calls the display_details method on the student1 object. This executes the code inside the display_details method for student1, printing its details**

Task Description#2 (Loops)

• Prompt AI to complete a function that prints the first 10 multiples of a number using a

loop.

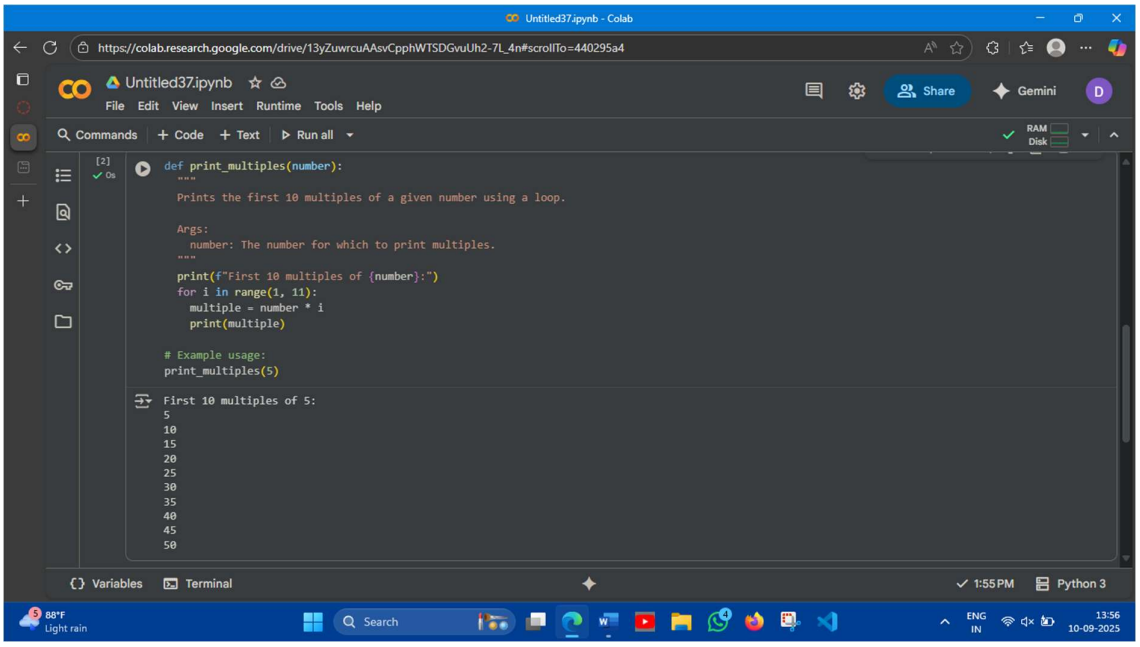
- Analyze the generated code
- Ask AI to generate code using other controlled looping

Expected Output#2

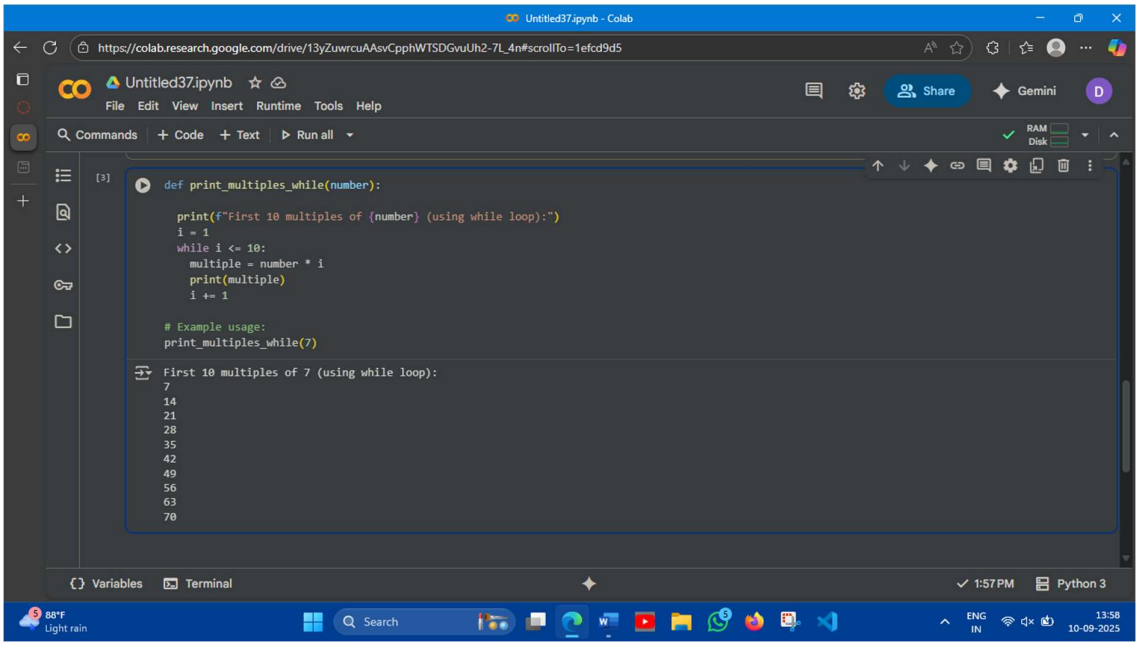
- Correct loop-based implementation

Prompt: generate code of function with loops that prints first 10 multiples of numbers

Screenshots:



Generate code using other controlled looping



Code Analysis:

- **def print_multiples_while(number):: This line defines a function named print_multiples_while that takes one argument, number.**
- **print(f"First 10 multiples of {number} (using while loop):"): This line prints a descriptive header indicating the output that follows.**
- **i = 1: This initializes a variable i to 1. This variable will act as a counter for the multiples.**
- **while i <= 10:: This is the while loop condition. The code block inside the while loop will continue to execute as long as the value of i is less than or equal to 10.**

- **multiple = number * i: Inside the loop, this line calculates the current multiple by multiplying the input number by the counter i.**
- **print(multiple): This line prints the calculated multiple.**
- **i += 1: This is a crucial step in a while loop. It increments the value of i by 1 in each iteration. This ensures that the loop will eventually terminate when i becomes greater than 10. If this line were missing, the loop would run indefinitely (an infinite loop).**
- **print_multiples_while(7): This line calls the print_multiples_while function with the argument 7, triggering the execution of the code within the function to print the first 10 multiples of 7.**

Task Description#3 (Conditional Statements)

- **Ask AI to write nested if-elif-else conditionals to classify age groups.**
- **Analyze the generated code**
- **Ask AI to generate code using other conditional statements**

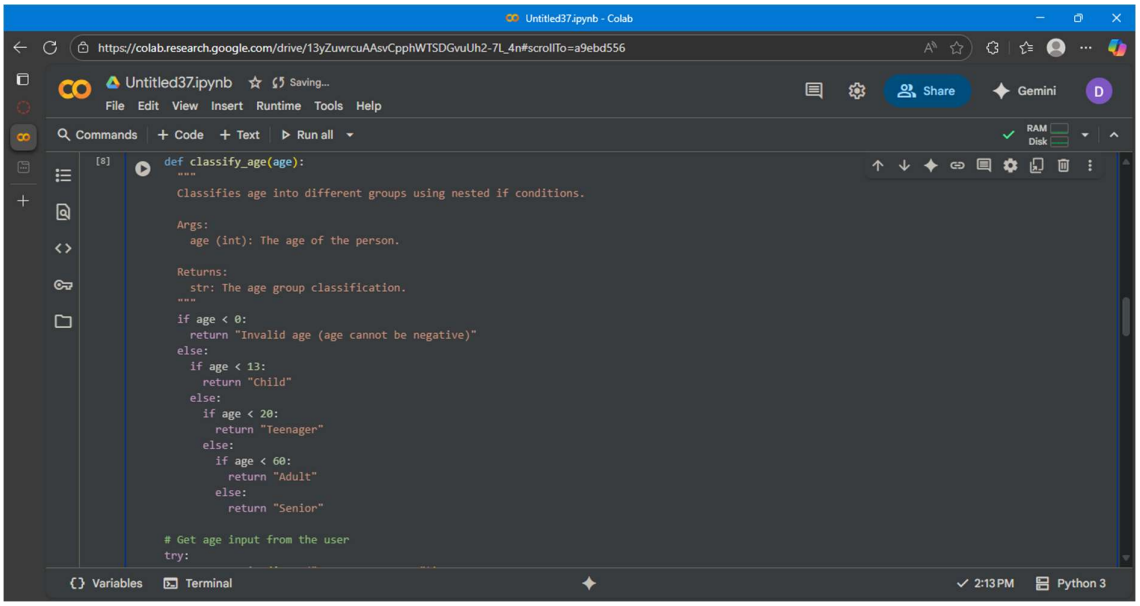
Expected Output#3

- **Age classification function with appropriate conditions and with explanation**

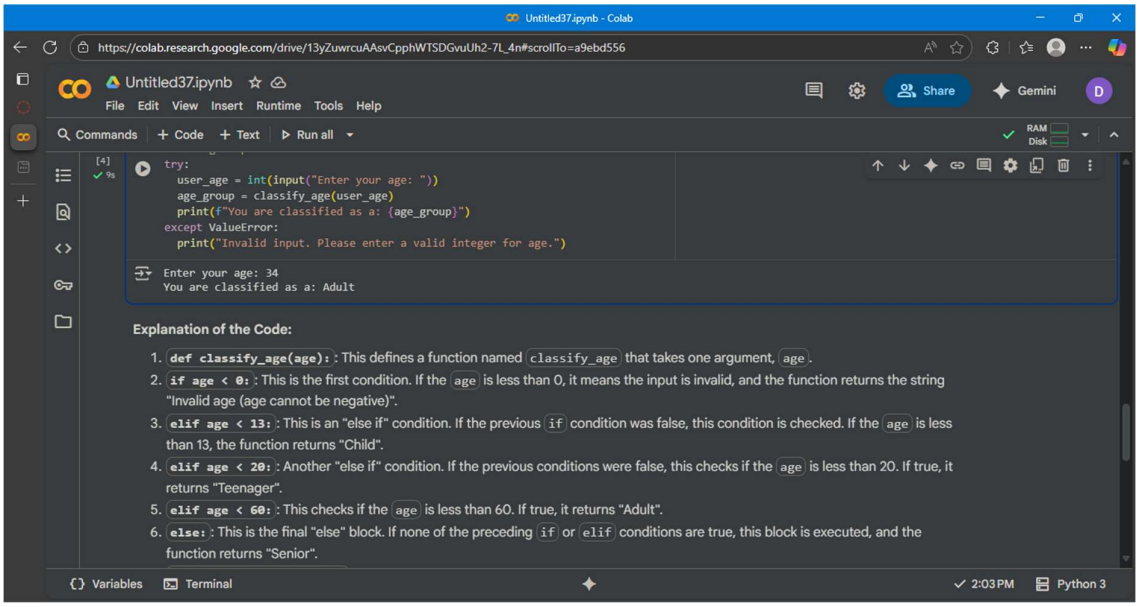
Prompt:

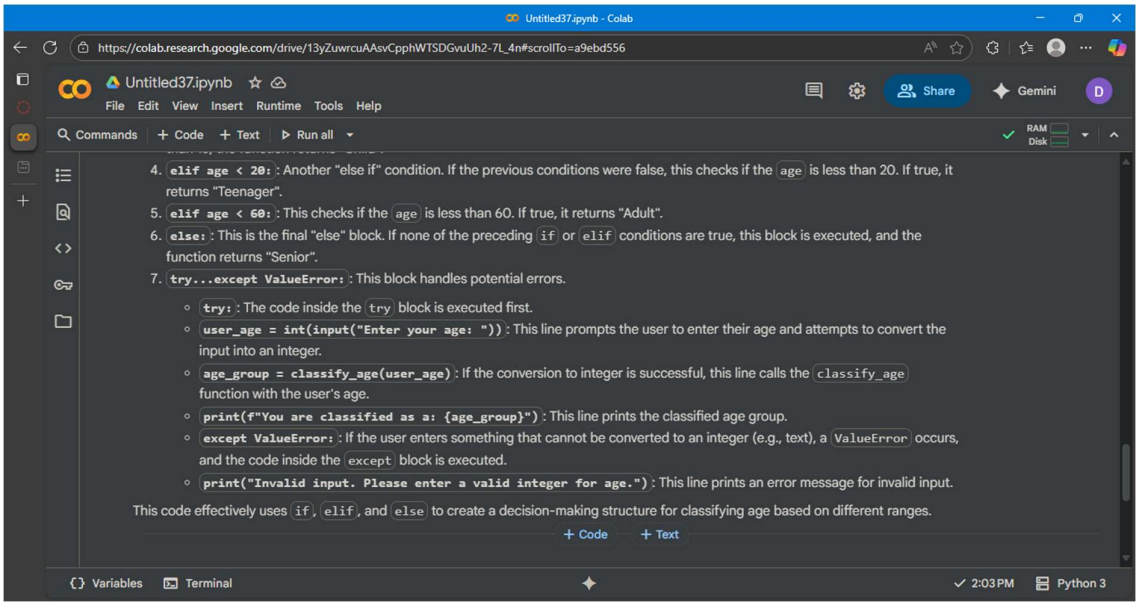
generate code for Age classification function with appropriate conditions and with explanation using if,elif,else conditionals to classify age groups with user given input

Screenshots:



Explaination:





This code defines a function classify_age that categorizes an input age. It uses if, elif, and else to check different age ranges. The try...except block handles potential invalid input from the user. It returns a string indicating the age group or an error message for negative/invalid ages.

Task Description#4 (For and While loops)

- **Generate a sum_to_n() function to calculate sum of first n numbers**
- **Analyze the generated code**
- **Get suggestions from AI with other controlled looping**

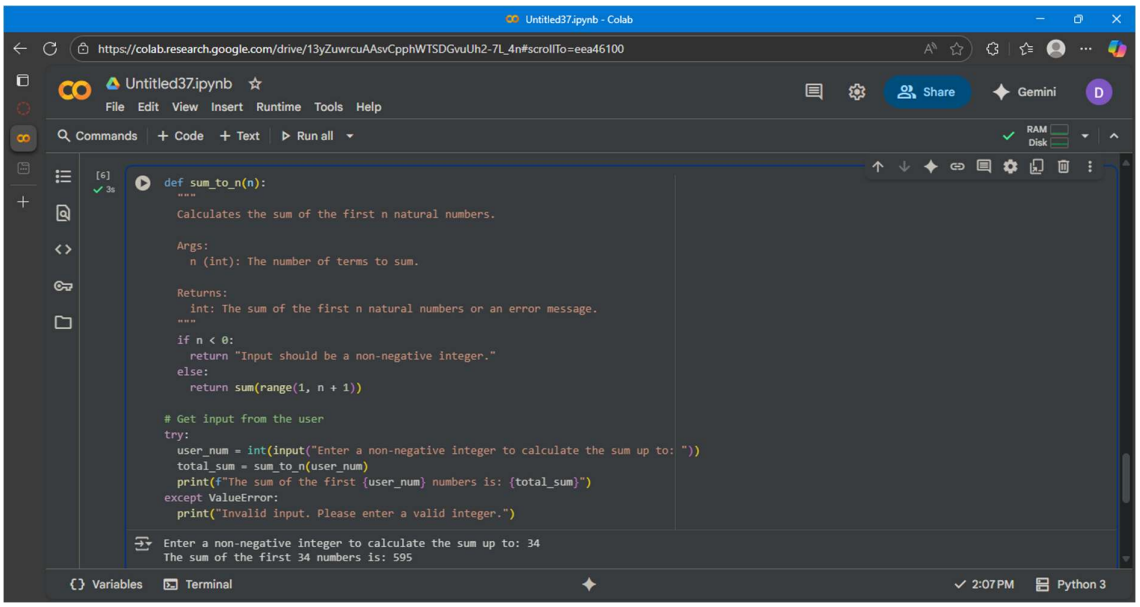
Expected Output#4

- **Python code with explanation**

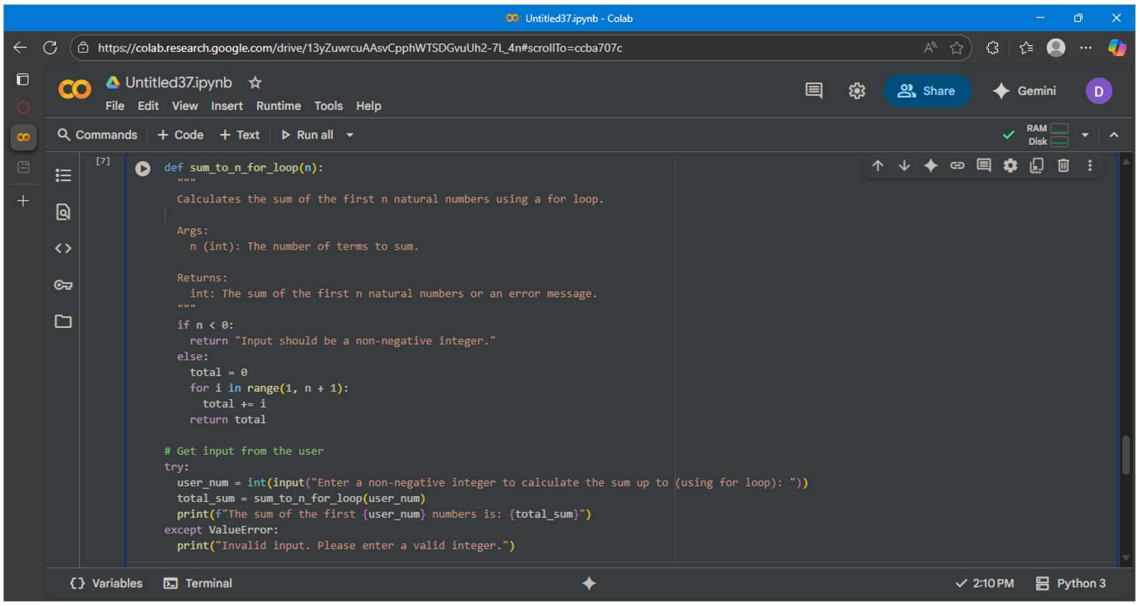
Prompt:

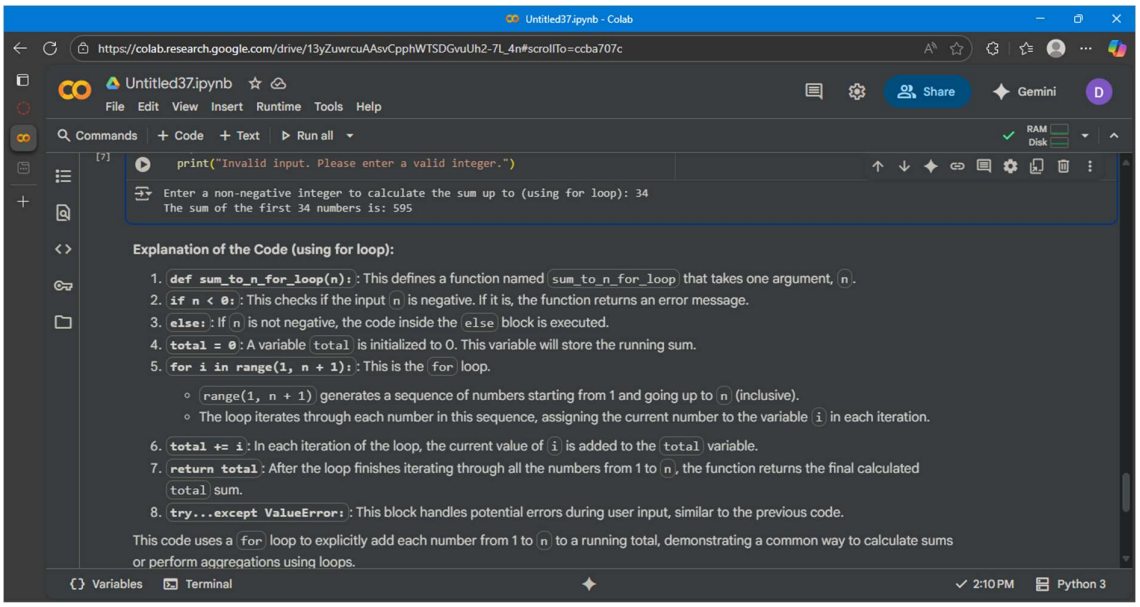
Generate a sum_to_n() function to calculate sum of first n numbers with user given input

Screenshots:



generate other controlled looping with explaination





Code analysis:

This function calculates the sum of the first n non-negative integers. It initializes a total to zero. A for loop iterates from 1 up to n. In each iteration, the current number is added to total. Finally, the accumulated total is returned

Task Description#5 (Class)

- **Use AI to build a BankAccount class with deposit, withdraw, and balance methods.**
- **Analyze the generated code**
- **Add comments and explain code**

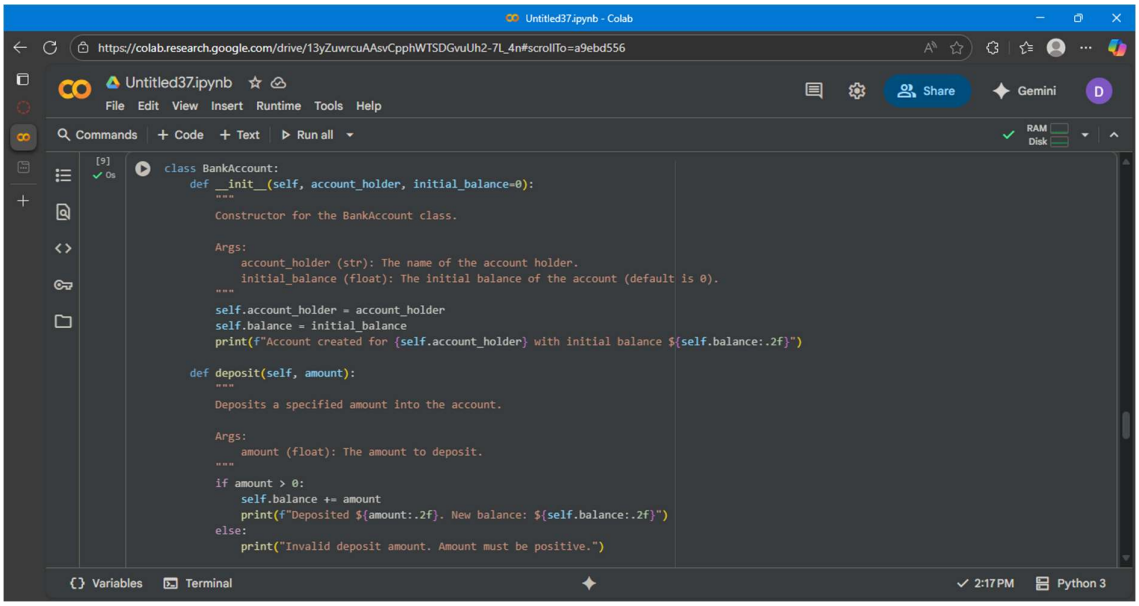
Expected Output#5

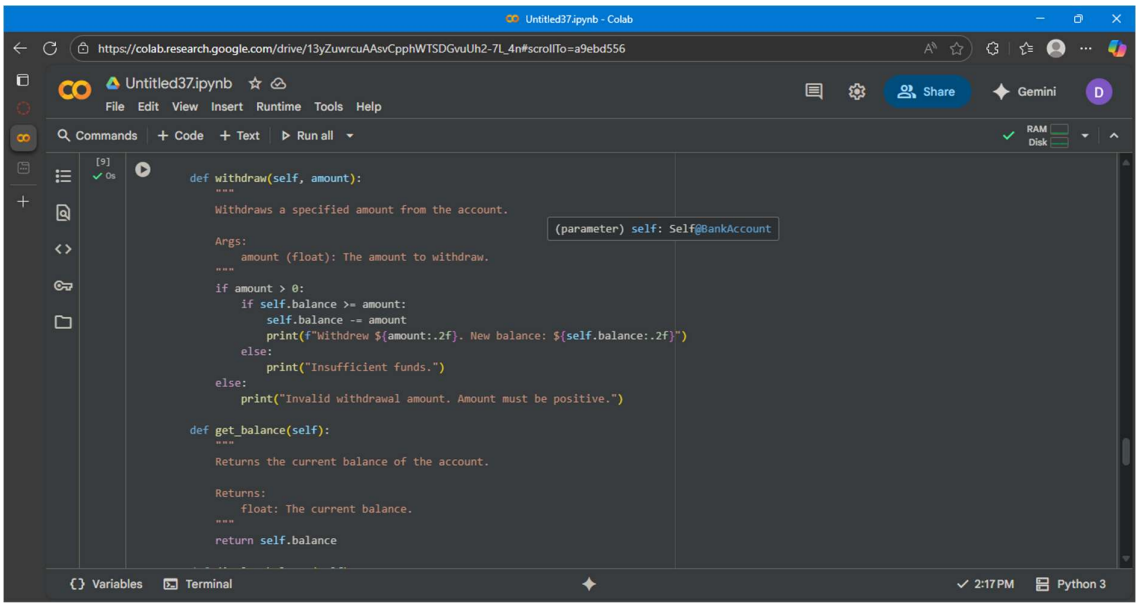
- **Python code with explanation**

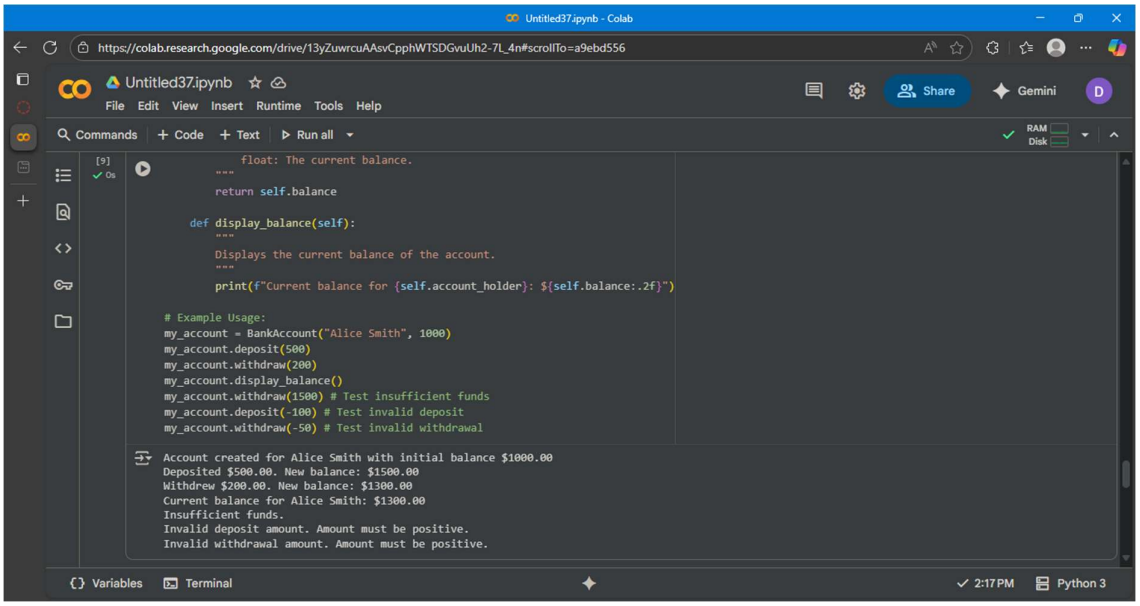
Prompt:

generate a python code for building a BankAccount class with deposit, withdraw, and balance methods with comments and explainations

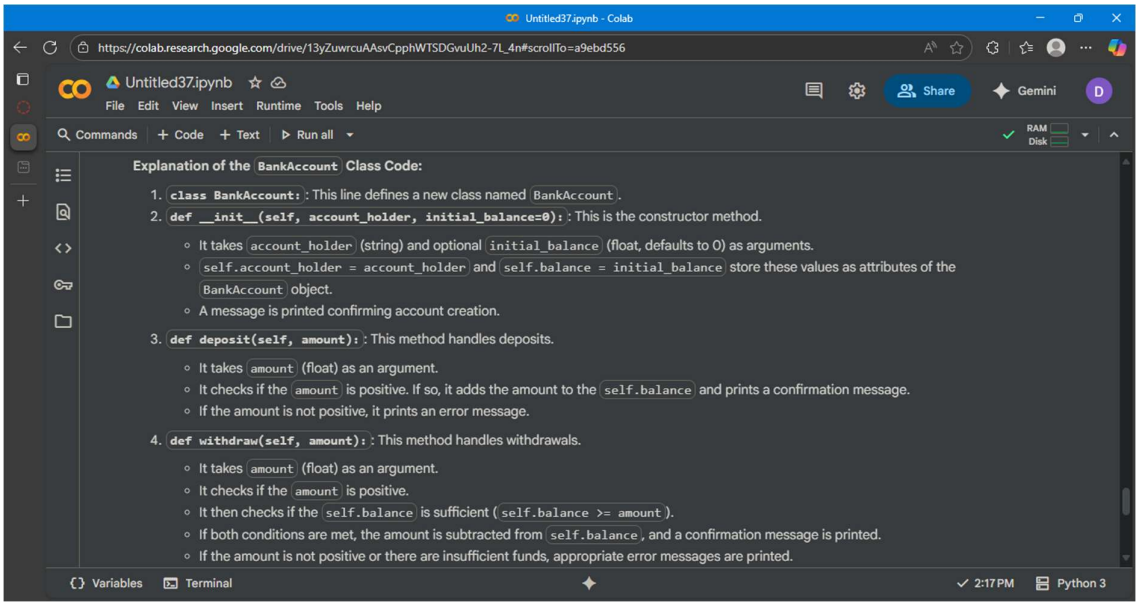
screenshots:

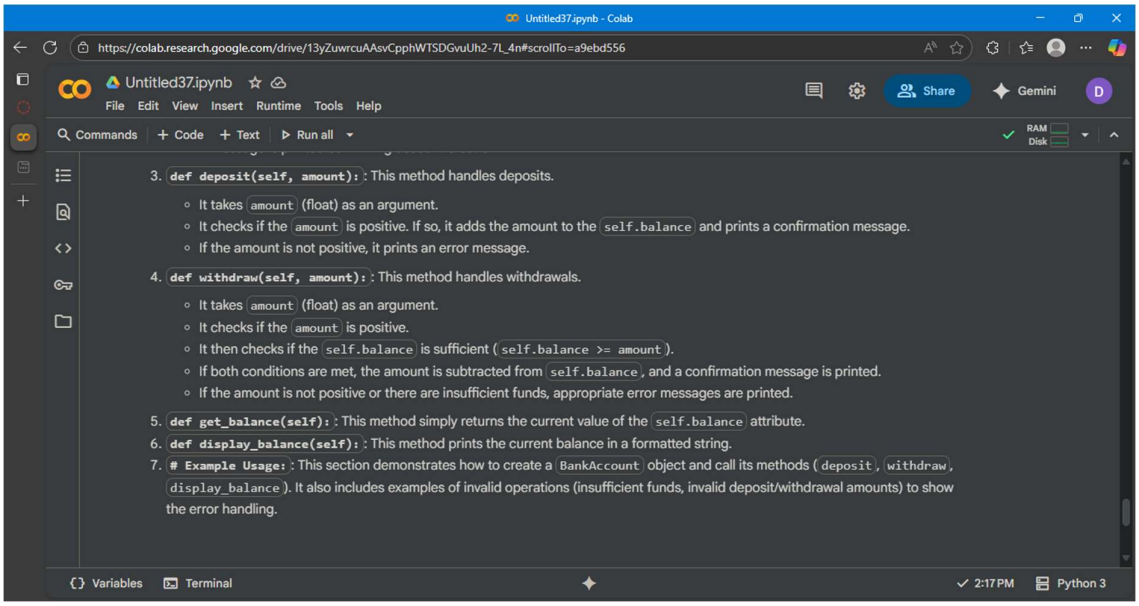

Untitled37.ipynb - Colab
https://colab.research.google.com/drive/13yZuwrcuAAsvCpphWTSDGvuUh2-7L_4n#scrollTo=a9ebd556
Untitled37.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM Disk
[9] 0s
class BankAccount:
def __init__(self, account_holder, initial_balance=0):
"""
Constructor for the BankAccount class.
Args:
account_holder (str): The name of the account holder.
initial_balance (float): The initial balance of the account (default is 0).
"""
self.account_holder = account_holder
self.balance = initial_balance
print(f"Account created for {self.account_holder} with initial balance ${self.balance:.2f}")
def deposit(self, amount):
"""
Deposits a specified amount into the account.
Args:
amount (float): The amount to deposit.
"""
if amount > 0:
self.balance += amount
print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
else:
print("Invalid deposit amount. Amount must be positive.")
Variables Terminal
2:17 PM Python 3


Untitled37.ipynb - Colab
https://colab.research.google.com/drive/13yZuwrcuAAsvCpphWTSDGvuUh2-7L_4n#scrollTo=a9ebd556
Untitled37.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM Disk
[9] 0s
def withdraw(self, amount):
"""
Withdraws a specified amount from the account.
(parameter) self: Self@BankAccount
Args:
amount (float): The amount to withdraw.
"""
if amount > 0:
if self.balance >= amount:
self.balance -= amount
print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
else:
print("Insufficient funds.")
else:
print("Invalid withdrawal amount. Amount must be positive.")
def get_balance(self):
"""
Returns the current balance of the account.
Returns:
float: The current balance.
"""
return self.balance
Variables Terminal
2:17 PM Python 3



Explaination:




Untitled37.ipynb - Colab
https://colab.research.google.com/drive/13yZuwrcuAAsvCpphWTSDGvuUh2-7L_4n#scrollTo=a9ebd556
Untitled37.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
3. def deposit(self, amount): : This method handles deposits.
It takes amount (float) as an argument.
It checks if the amount is positive. If so, it adds the amount to the self.balance and prints a confirmation message.
If the amount is not positive, it prints an error message.
4. def withdraw(self, amount): : This method handles withdrawals.
It takes amount (float) as an argument.
It checks if the amount is positive.
It then checks if the self.balance is sufficient (self.balance >= amount).
If both conditions are met, the amount is subtracted from self.balance , and a confirmation message is printed.
If the amount is not positive or there are insufficient funds, appropriate error messages are printed.
5. def get_balance(self): : This method simply returns the current value of the self.balance attribute.
6. def display_balance(self): : This method prints the current balance in a formatted string.
7. # Example Usage: : This section demonstrates how to create a BankAccount object and call its methods (deposit , withdraw , display_balance). It also includes examples of invalid operations (insufficient funds, invalid deposit/withdrawal amounts) to show the error handling.
Variables
Terminal
2:17 PM
Python 3