

Program	:B.tech(CSE)
Specialization	:AIML
Course Title	:AI Assisted Coding
Course Code	:24CS002PC215
Semester	:3 <sup>rd</sup> semester
Academic Session	:2025-2026
Name of Student	: <b><i>Kurapati Akshitha</i></b>
Enrollment No.	:2403A52021
Batch No.	:02
Date	:10/09/2025

---

## #LAB ASSIGNMENT

---

### #Task Description -1:

Use AI to complete a Student class with attributes and a method.

Check output

Analyze the code generated by AI tool.

### #PROMPT:

Write a Python class called Student with attributes like name, age, and grade.  
Add a method to display the student's information. After generating the code,  
check the output and analyze the logic for clarity and correctness.

### #QUESTION:

The screenshot shows a code editor with a Python class named `Student`. The class has an `__init__` method that takes `name`, `age`, `grade`, and `hall_ticket_number` as arguments and assigns them to instance variables. It also has a `display_info` method that prints the student's details. Below the class definition, there is a script that takes user input for these details, creates a `Student` object, and calls `display_info`. The output of the script is shown in the terminal: `Enter student name: Deekshitha`, `Enter student age: 18`, `Enter student grade: 9`, `Enter hall ticket number: 32`, followed by the printed student information.

Overlaid on the right is a chat window from Gemini AI. The prompt is: "Write a Python class called Student with attributes like name, age, and grade. Add a method to display the student's information. After generating the code, check the output and analyze the logic for clarity and correctness." The response from Gemini is: "Happy to help! I will now generate the code for the Student class as you described. After that, I will analyze the output and logic. Great! The code for the Student class has been executed successfully. The output shows the information for both student1 and student2 is". At the bottom of the chat window, there is a text input field with the placeholder "What can I help you build?" and a "Send" button.

## #CODE:

This screenshot shows the same Python code as the first image, but without the chat window overlay. The code defines a `Student` class with an `__init__` method and a `display_info` method. Below the class, a script prompts the user for student details, creates a `Student` object, and calls the `display_info` method. The code is as follows:

```
class Student:
    """Represents a student with name, age, grade, and hall ticket number."""
    def __init__(self, name, age, grade, hall_ticket_number):
        self.name = name
        self.age = age
        self.grade = grade
        self.hall_ticket_number = hall_ticket_number

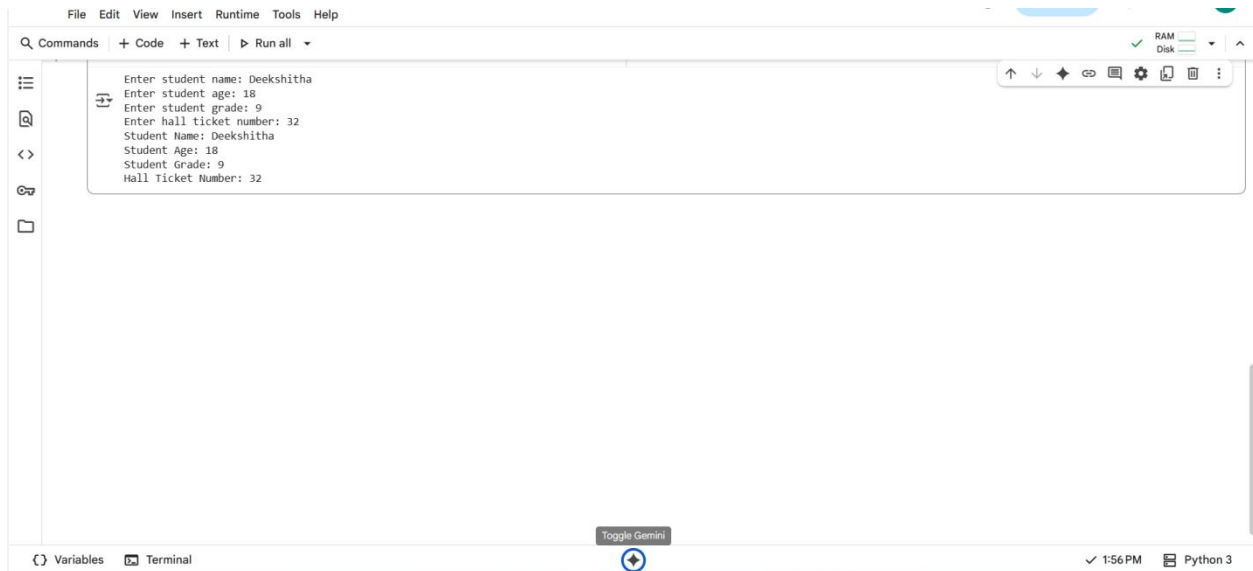
    def display_info(self):
        """Displays the student's information."""
        print(f"Student Name: {self.name}")
        print(f"Student Age: {self.age}")
        print(f"Student Grade: {self.grade}")
        print(f"Hall Ticket Number: {self.hall_ticket_number}")

# Get input from the user
name = input("Enter student name: ")
age = int(input("Enter student age: "))
grade = input("Enter student grade: ")
hall_ticket_number = input("Enter hall ticket number: ")

# Create a Student object with user input
user_student = Student(name, age, grade, hall_ticket_number)

# Display the user's student information
user_student.display_info()
```

## #OUTPUT:



## #COMMENT:

### Class Definition:

- `class Student:` : This line defines a new class named `Student`. Classes are blueprints for creating objects (like individual students in this case).
- `"""Represents a student with name, age, grade, and hall ticket number."""` : This is a docstring, which explains what the class does.
- `__init__(self, name, age, grade, hall_ticket_number):` : This is the constructor method. It's called when you create a new `Student` object.
  - `self` : Refers to the instance of the class being created.
  - `name, age, grade, hall_ticket_number` : These are parameters that you pass when creating a `Student` object.
  - `self.name = name` : This line assigns the value of the `name` parameter to the `name` attribute of the object. The same applies to `age`, `grade`, and `hall_ticket_number`.

### Methods:

- `display_info(self):` : This is a method (a function within a class) that displays the student's information.
  - `self` : Again, refers to the instance of the class.
  - `print(f"...")` : These lines print the student's details using f-strings for easy formatting. They access the attributes using `self.attribute_name`.

### User Input and Object Creation:

- `name = input("Enter student name: ")` : This line prompts the user to enter the student's name and stores it in the `name` variable. The same applies to `age`, `grade`, and `hall_ticket_number`. Note that `age` is converted to an integer using `int()`.
- `user_student = Student(name, age, grade, hall_ticket_number)` : This line creates a new `Student` object using the input provided by the user and assigns it to the `user_student` variable.
- `user_student.display_info()` : This line calls the `display_info` method on the `user_student` object to display the information entered by the user.

## #Task Description -2:

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.

## #PROMPT:

Write a Python function that prints the first 10 multiples of a given number using a loop. Then rewrite the function using a different loop type and compare both versions.

## #QUESTION:

The screenshot shows a code editor with a sidebar on the left containing icons for file explorer, search, and other tools. The main area displays a task description and an AI-generated response.

**User Input and Object Creation:**

- `name = input("Enter student name: ")`: This line prompts the user to enter the student's name and stores it in the `name` variable. The same applies to `age`, `grade`, and `hall_ticket_number`. Note that `age` is converted to an integer using `int()`.
- `user_student = Student(name, age, grade, hall_ticket_number)`: This line creates a new `Student` object using the input provided by the user and assigns it to the `user_student` variable.
- `user_student.display_info()`: This line calls the `display_info` method on the `user_student` object to display the information entered by the user.

**Task**

Write a Python function that prints the first 10 multiples of a given number using a loop and compare both versions.

**Generate function with `for` loop**

**Subtask:**

Create a Python function that takes a number as input and prints its first 10 multiples using a `for` loop and the `range()` function.

**Reasoning:** Define the function `print_multiples_of_number`.

**AI Response:**

Write a Python function that prints the first 10 multiples of a given number using a loop. Then rewrite the function using a different loop type and compare both versions.

Happy to help! Here is a plan to create a Python function that prints the first 10 multiples of a given number using different loops and compare them.

- **Generate function with `for` loop:** Create a Python function that takes a number as input and prints its first 10 multiples using a `for` loop and the `range()` function.

What can I help you build?

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

## #CODE:

Commands + Code + Text ▶ Run all

Task

Write a Python function that prints the first 10 multiples of a given number using a loop. Then rewrite the function using a different loop type and compare both versions.

Generate function with for loop

Subtask:

Create a Python function that takes a number as input and prints its first 10 multiples using a for loop and the range() function.

Reasoning: Define the function print\_multiples\_for using a for loop and range() to print the first 10 multiples of the input number.

```
[11] ✓ 16s
def print_multiples_for(number):
    """Prints the first 10 multiples of a given number using a for loop.

    Args:
        number: The number for which to print multiples.
    """
    for i in range(1, 11):
        multiple = number * i
        print(multiple)

# Get input from the user
```

Variables Terminal

Executing (27s) Python 3

Untitled12.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```
[11] ✓ 16s
# Get input from the user
num_for = int(input("Enter a number to print its first 10 multiples (for loop): "))
# Example usage:
print_multiples_for(num_for)
```

#OUTPUT:

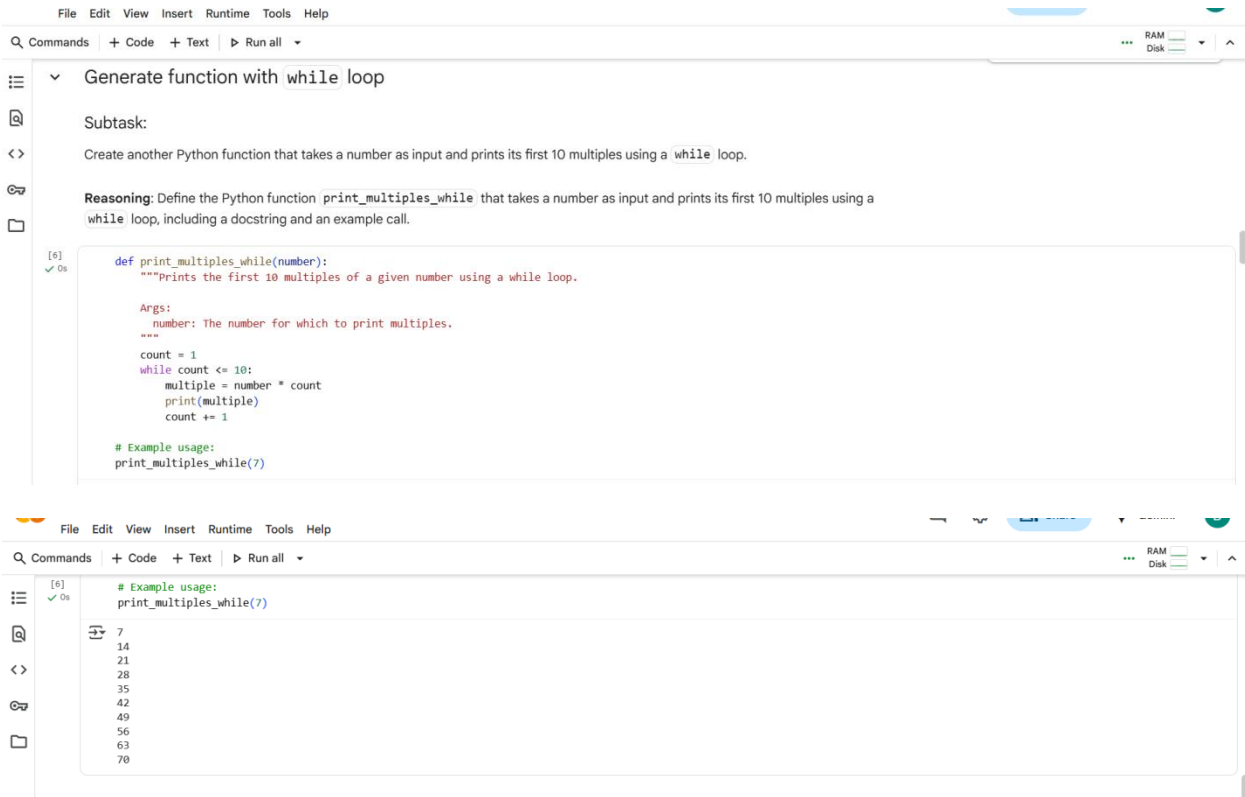
Commands + Code + Text ▶ Run all

```
[11] ✓ 16s
# Get input from the user
num_for = int(input("Enter a number to print its first 10 multiples (for loop): "))
# Example usage:
print_multiples_for(num_for)
```

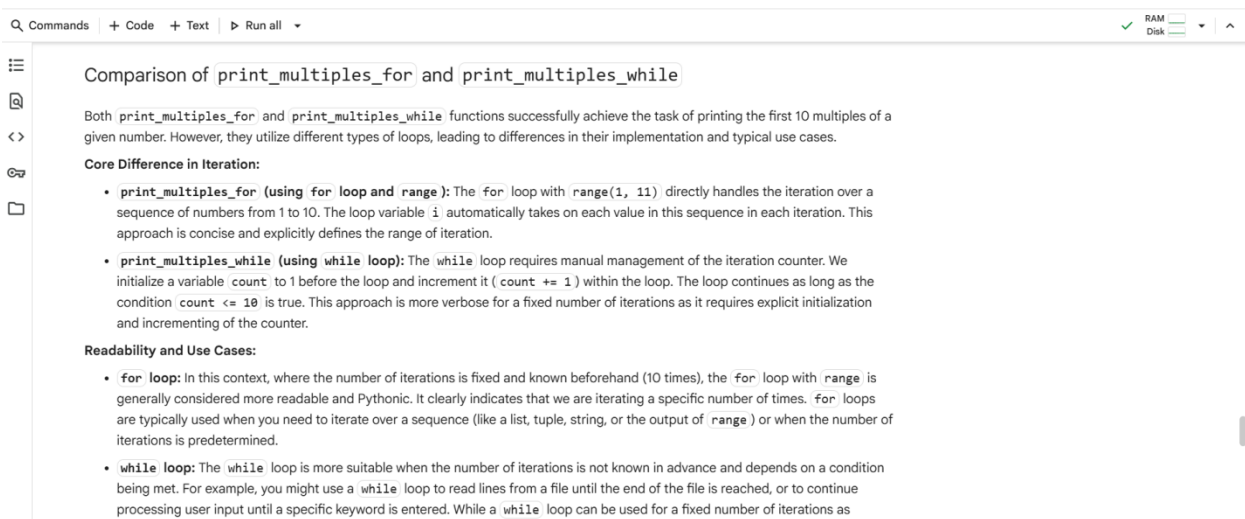
Enter a number to print its first 10 multiples (for loop): 8

8  
16  
24  
32  
40  
48  
56  
64  
72  
80

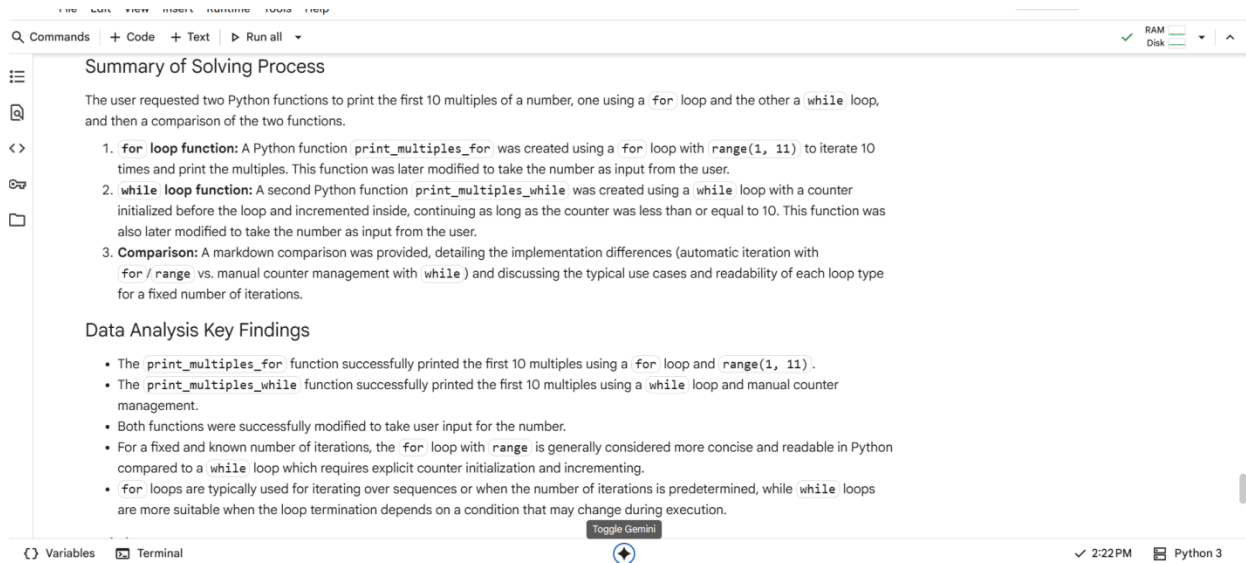
#CODE:



## #COMPARISON



## #SUMMARY:



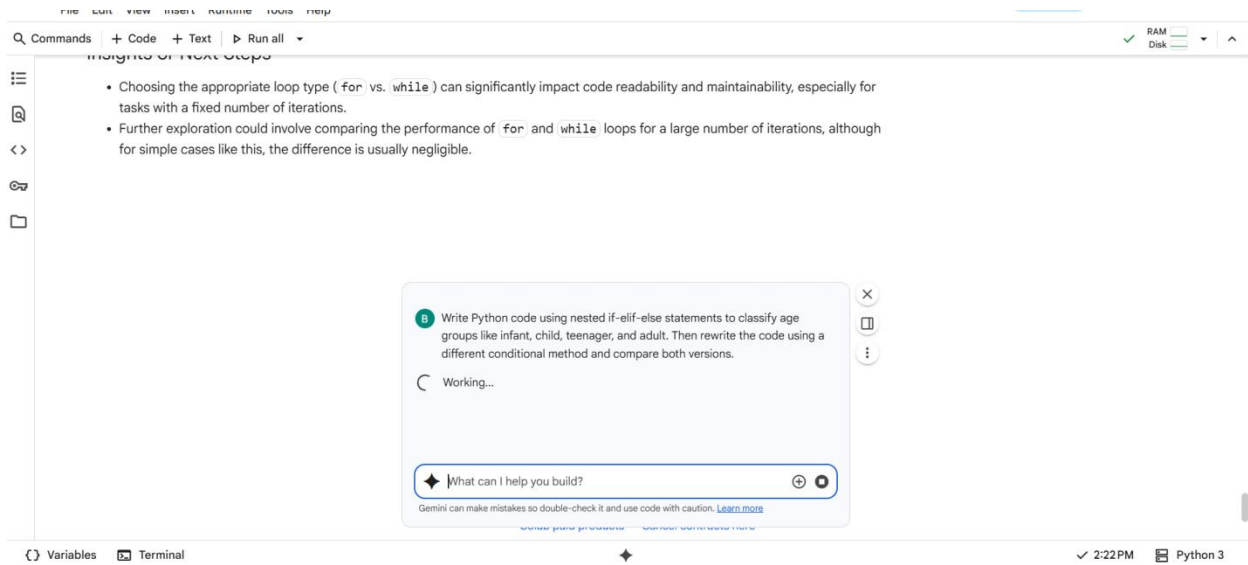
## #Task Description -3:

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. write perfect and simple prompt for this question.

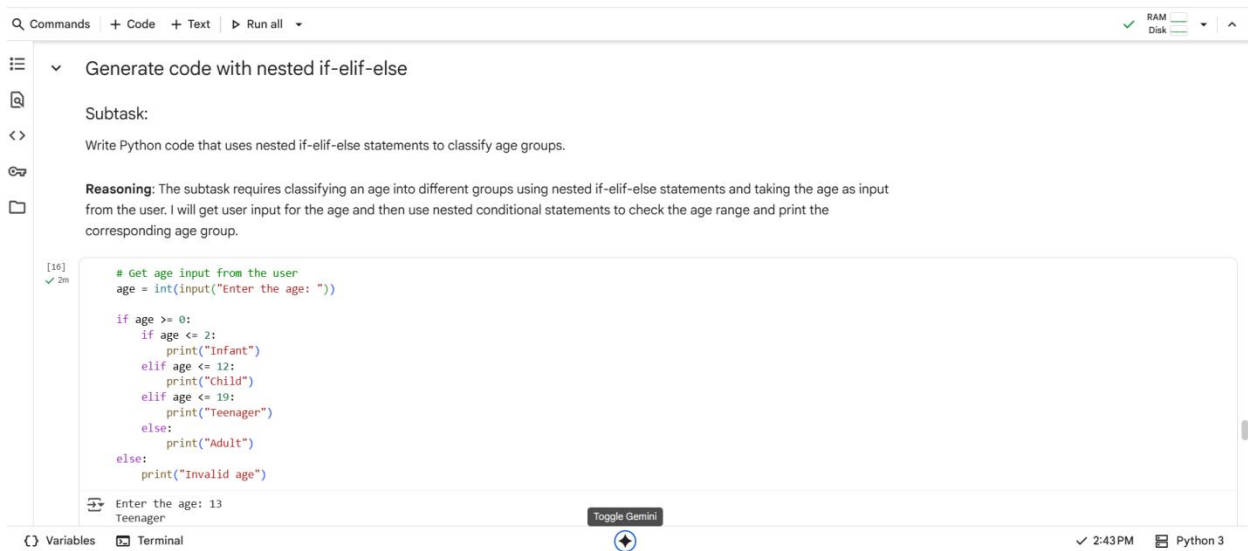
### #PROMPT:

Write Python code using nested if-elif-else statements to classify age groups like infant, child, teenager, and adult. Then rewrite the code using a different conditional method and compare both versions.

### #QUESTION:



## #CODE with OUTPUT:





File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

RAM Disk

Generate code with a different conditional method

Subtask:

Rewrite the age group classification code using a different conditional method (e.g., chained if-elif-else, or a dictionary/list-based approach).

Reasoning: The subtask requires rewriting the age classification code using a chained if-elif-else structure and taking the age as input from the user. I will get user input for the age and then implement the chained conditional statements to classify the age based on the provided criteria and print the result.

[17] ✓ 7s

# Get age input from the user  
age = int(input("Enter the age: "))  
  
if age >= 0 and age <= 2:  
 print("Infant")  
elif age >= 3 and age <= 12:  
 print("Child")  
elif age >= 13 and age <= 19:  
 print("Teenager")  
elif age >= 20:  
 print("Adult")  
else:  
 print("Invalid age")

Enter the age: 18  
Teenager

Toggle Gemini

Variables Terminal 2:43 PM Python 3

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

RAM Disk

Compare the code versions

Subtask:

Compare the two versions of the age classification code, discussing their readability, efficiency, and suitability for different scenarios.

Reasoning: Create a markdown cell to compare the two age classification methods as requested in the subtask.

Comparison of Age Classification Methods

We have explored two ways to classify age groups using conditional statements in Python: nested `if-elif-else` and chained `if-elif-else`. Both methods successfully classify an age into "Infant", "Child", "Teenager", or "Adult" based on predefined age ranges, and handle invalid age input.

Nested `if-elif-else` (as in Cell 0a80b51d):

[18] ✓ 15s

# Get age input from the user  
age = int(input("Enter the age: "))  
  
if age >= 0 and age <= 2:  
 print("Infant")  
elif age >= 3 and age <= 12:  
 print("Child")  
elif age >= 13 and age <= 19:  
 print("Teenager")  
elif age >= 20:  
 print("Adult")

Toggle Gemini

Variables Terminal 2:43 PM Python 3

Commands | + Code | + Text | ▶ Run all

Reasoning: Create a markdown cell to compare the two age classification methods as requested in the subtask.

### Comparison of Age Classification Methods

We have explored two ways to classify age groups using conditional statements in Python: nested `if-elif-else` and chained `if-elif-else`. Both methods successfully classify an age into "Infant", "Child", "Teenager", or "Adult" based on predefined age ranges, and handle invalid age input.

**Nested `if-elif-else` (as in Cell 0a80b51d):**

```
[18] ✓ 15s
# Get age input from the user
age = int(input("Enter the age: "))

if age >= 0 and age <= 2:
    print("Infant")
elif age >= 3 and age <= 12:
    print("Child")
elif age >= 13 and age <= 19:
    print("Teenager")
elif age >= 20:
    print("Adult")
else:
    print("Invalid age")
```

Enter the age: 26  
Adult

Variables | Terminal | 2:43 PM | Python 3

## #SUMMARY:

File | Edit | View | Insert | Runtime | Tools | Help

Commands | + Code | + Text | ▶ Run all

### Summary:

#### Data Analysis Key Findings

- The nested `if-elif-else` code classified an age of 25 as "Adult".
- The chained `if-elif-else` code also classified an age of 25 as "Adult".
- The chained `if-elif-else` structure is considered more readable for sequential condition checks like age ranges compared to nested structures for this specific task.
- Efficiency between the two methods is negligible for this task.
- Nested structures are more suitable for hierarchical conditions, while chained structures are better for sequential and mutually exclusive conditions.

#### Insights or Next Steps

- For tasks involving sequential and mutually exclusive range checks, the chained `if-elif-else` structure is generally preferred for its readability and maintainability.
- Consider using dictionaries or other data structures for more complex classification tasks with many categories to improve scalability and readability further.

Variables | Terminal | 2:43 PM | Python 3

## #Task Description -4:

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.

## #PROMPT:

Write a Python function `sum_to_n(n)` that calculates the sum of the first `n` natural numbers using a loop. Then rewrite the function using a different type of loop and compare both versions.

## #QUESTION:

The screenshot shows a code editor interface with a task description and a chat window. The task is to write a Python function `sum_to_n(n)` that calculates the sum of the first `n` natural numbers using a loop, and then rewrite it using a different type of loop. The chat window shows a Gemini AI assistant responding to the task.

**Generate function with a loop**

**Subtask:**

Create a Python function `sum_to_n` that takes an integer `n` as input from the user and calculates the sum of the first `n` natural numbers using a loop (e.g., a `for` loop).

**Reasoning:** Define the Python function `sum_to_n_for` that takes user input for `n` and calculates the sum of the first `n` natural numbers using a `for` loop and `range()`. Include a docstring and an example call.

**Generate function with a different loop**

**Subtask:**

Rewrite the `sum_to_n` function using a different type of loop (e.g., a `while` loop).

**Reasoning:** Define the Python function `sum_to_n_while` that takes user input for `n` and calculates the sum of the first `n` natural numbers using a `while` loop. Include a docstring and an example call.

**Chat Window:**

Write a Python function `sum_to_n(n)` that calculates the sum of the first `n` natural numbers using a loop. Then rewrite the function using a different type of loop and compare both versions.

Happy to help! I will create a plan to write a Python function that calculates the sum of the first `n` natural numbers using different loops and compare them.

- **Generate function with a loop:** Create a Python function `sum_to_n` that takes an integer `n` as input and calculates the sum of the first `n` natural numbers using a `for` loop.

What can I help you build?

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

```
[1] def sum_to_n_for(n):  
    """Calculates the sum of the first n natural numbers using a for loop.  
    29%
```

## #CODE

The screenshot shows the same code editor interface, but with the code for the `while` loop version of the `sum_to_n` function. The chat window is closed.

**Generate function with a loop**

**Subtask:**

Create a Python function `sum_to_n` that takes an integer `n` as input from the user and calculates the sum of the first `n` natural numbers using a loop (e.g., a `for` loop).

**Reasoning:** Define the Python function `sum_to_n_for` that takes user input for `n` and calculates the sum of the first `n` natural numbers using a `for` loop and `range()`. Include a docstring and an example call.

**Generate function with a different loop type**

**Subtask:**

Rewrite the `sum_to_n` function using a different type of loop (e.g., a `while` loop).

**Reasoning:** Define the Python function `sum_to_n_while` that takes user input for `n` and calculates the sum of the first `n` natural numbers using a `while` loop. Include a docstring and an example call.

```
[1] def sum_to_n_while(n):  
    """Calculates the sum of the first n natural numbers using a while loop.  
    29%
```

Reasoning: Define the Python function `sum_to_n_while` that takes user input for `n` and calculates the sum of the first `n` natural numbers using a `while` loop. Include a docstring and an example call.

```
[1] 29s def sum_to_n_while(n):  
    """Calculates the sum of the first n natural numbers using a while loop.  
  
    Args:  
        n: The number of natural numbers to sum up to.  
  
    Returns:  
        The sum of the first n natural numbers.  
    """  
    total_sum = 0  
    count = 1  
    while count <= n:  
        total_sum += count  
        count += 1  
    return total_sum  
  
# Get input from the user  
num_n_while = int(input("Enter a positive integer (while loop): "))  
  
# Calculate and print the sum  
if num_n_while > 0:  
    result_while = sum_to_n_while(num_n_while)  
    print(f"The sum of the first {num_n_while} natural numbers is: {result_while}")  
else:  
    print("Please enter a positive integer.")
```

Toggle Gemini

Variables Terminal Executing (0s) Python 3

## #OUTPUT:

File Edit View Insert Runtime Tools Help

```
[1] 29s # Get input from the user  
num_n_while = int(input("Enter a positive integer (while loop): "))  
  
# Calculate and print the sum  
if num_n_while > 0:  
    result_while = sum_to_n_while(num_n_while)  
    print(f"The sum of the first {num_n_while} natural numbers is: {result_while}")  
else:  
    print("Please enter a positive integer.")
```

Enter a positive integer (while loop): 10  
The sum of the first 10 natural numbers is: 55

Toggle Gemini

Variables Terminal Executing (0s) Python 3

## Task Description -5:

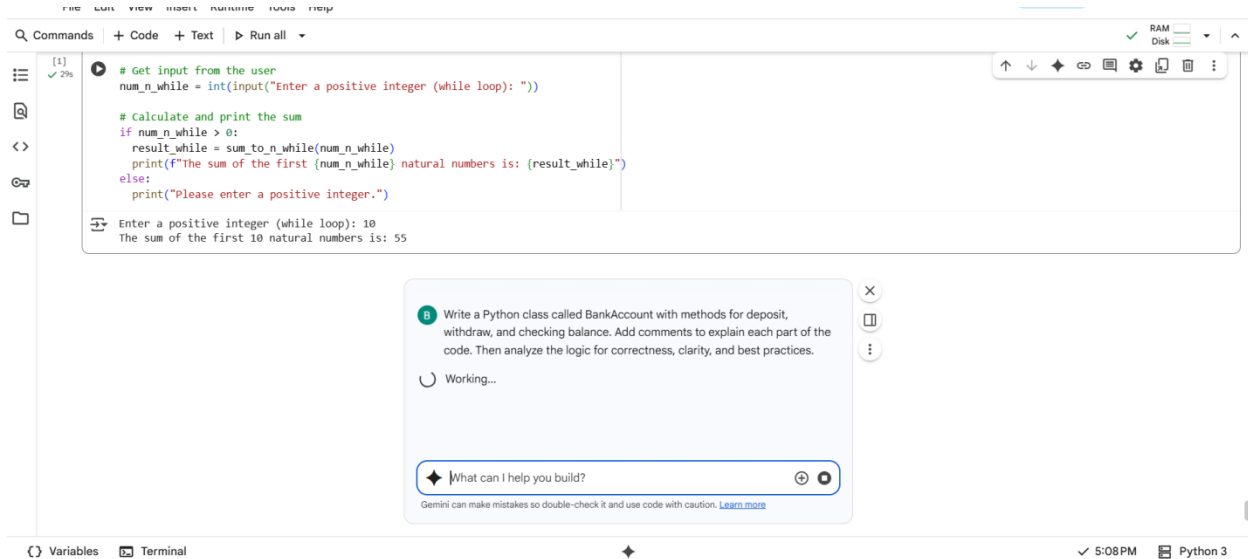
Use AI to build a `BankAccount` class with `deposit`, `withdraw`, and `balance` methods.

- Analyze the generated code
- Add comments and explain code.

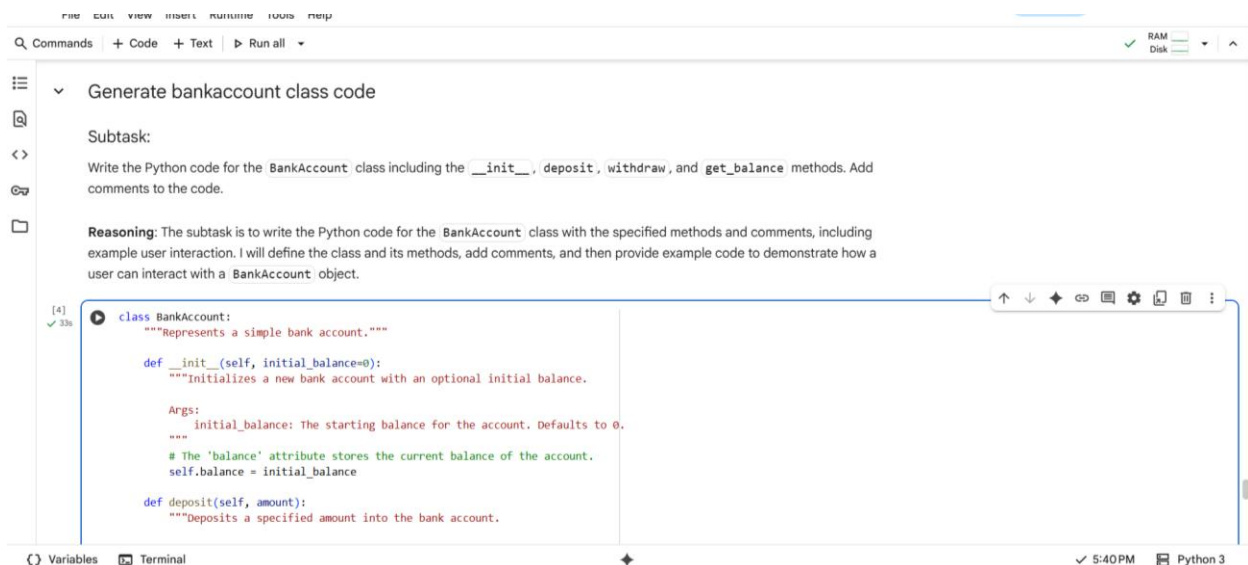
## #PROMPT:

Write a Python class called `BankAccount` with methods for `deposit`, `withdraw`, and checking `balance`. Add comments to explain each part of the code. Then analyze the logic for correctness, clarity, and best practices.

## #QUESTION:



## #CODE



```
Commands | + Code | + Text | ▶ Run all | RAM | Disk | ✓ | ^
[4] | ✓ 330 |
Args:
    amount: The amount to be deposited.
"""
if amount > 0:
    self.balance += amount
    print(f"Deposited: ${amount}. New balance: ${self.balance}")
else:
    print("Deposit amount must be positive.")

def withdraw(self, amount):
    """Withdraws a specified amount from the bank account if sufficient funds are available.

    Args:
        amount: The amount to be withdrawn.
    """
    # Check if the withdrawal amount is positive and if there are sufficient funds.
    if amount > 0:
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew: ${amount}. New balance: ${self.balance}")
        else:
            print("Insufficient funds.")
    else:
        print("Withdrawal amount must be positive.")

def get_balance(self):
    """Returns the current balance of the bank account.

    Returns:
        The current balance.
    """
```

Variables | Terminal | 5:40 PM | Python 3

```
Commands | + Code | + Text | ▶ Run all | RAM | Disk | ✓ | ^
[4] | ✓ 330 |
Returns:
    The current balance.
"""
return self.balance

# Example User Interaction:
# Create a bank account
my_account = BankAccount(1000)

# Get user input for deposit
deposit_amount = float(input("Enter amount to deposit: "))
my_account.deposit(deposit_amount)

# Get user input for withdrawal
withdraw_amount = float(input("Enter amount to withdraw: "))
my_account.withdraw(withdraw_amount)

# Check balance
print(f"Current balance: ${my_account.get_balance()}")
```

## #OUTPUT:

```
File Edit View Insert Runtime Tools Help
Commands | + Code | + Text | ▶ Run all | RAM | Disk | ✓ | ^
[4] | ✓ 330 |
# Check balance
print(f"Current balance: ${my_account.get_balance()}")

Enter amount to deposit: 20000
Deposited: $20000.0. New balance: $21000.0
Enter amount to withdraw: 11000
Withdraw: $11000.0. New balance: $10000.0
Current balance: $10000.0
```

## #EXPLANATION:

1. **# Example User Interaction:** This is a comment indicating that the following lines are an example of how a user would interact with the `BankAccount` class.
2. **# Create a bank account:** Another comment explaining the next line.
3. **my\_account = BankAccount(1000):** This line creates an *instance* of the `BankAccount` class. Think of `my_account` as a specific bank account object. When `BankAccount(1000)` is called, the `__init__` method within the `BankAccount` class is executed, setting the initial balance of this new account to 1000.
4. **# Get user input for deposit:** Comment for the deposit part.
5. **deposit\_amount = float(input("Enter amount to deposit: ")):** This line prompts the user to type in a value after displaying the message "Enter amount to deposit: ". The `input()` function reads whatever the user types as a string. `float()` then converts that string into a floating-point number (a number with a decimal point), and this value is stored in the variable `deposit_amount`.
6. **my\_account.deposit(deposit\_amount):** This line calls the `deposit` method on the `my_account` object. The value stored in `deposit_amount` (the number the user entered) is passed as the `amount` argument to the `deposit` method.

7. `# Get user input for withdrawal`: Comment for the withdrawal part.
8. `withdraw_amount = float(input("Enter amount to withdraw:"))`: Similar to the deposit input, this line prompts the user for a withdrawal amount, reads their input as a string, converts it to a floating-point number, and stores it in the `withdraw_amount` variable.
9. `my_account.withdraw(withdraw_amount)`: This line calls the `withdraw` method on the `my_account` object, passing the user's withdrawal amount to the method.
10. `# Check balance`: Comment for the balance check.
11. `print(f"Current balance: ${my_account.get_balance()}")`: This line first calls the `get_balance()` method on the `my_account` object. This method returns the current value of the `balance` attribute. The `print()` function then displays the text "Current balance: \$" followed by the balance value returned by `get_balance()`. The `f""` is an f-string, which allows you to embed the value of variables directly within the string.

THANK YOU



