

AI Assistant Coding

Assignment-6.3

Name:V.Sruthi

Ht.no:2303A51308

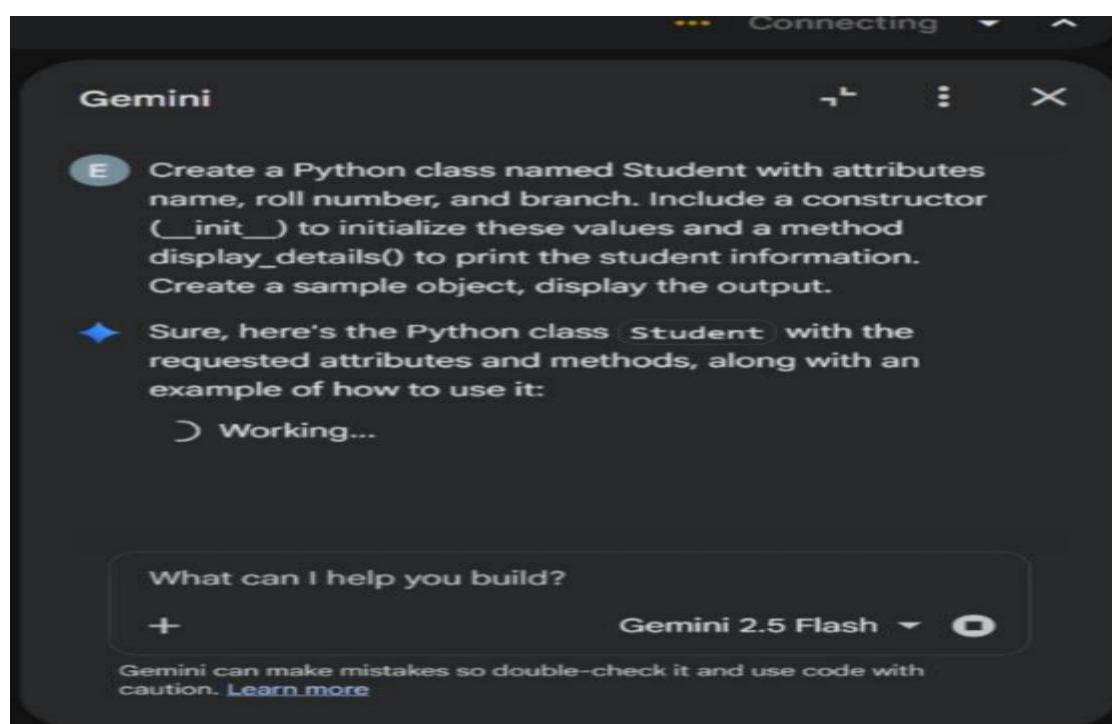
Batch:05

Task-01:

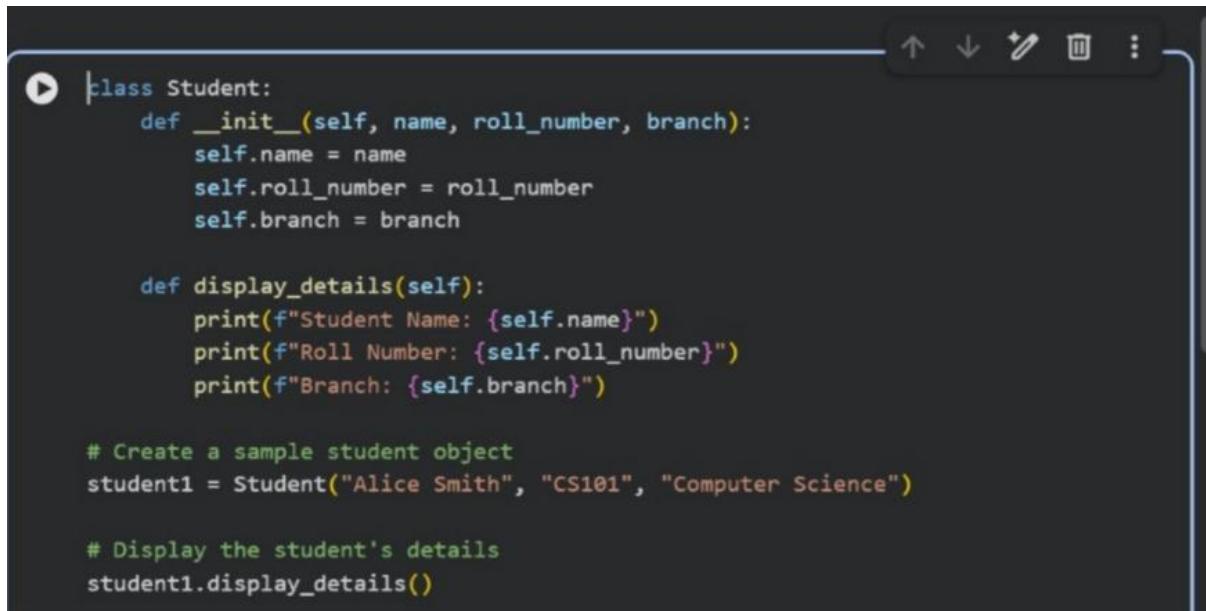
Classes (Student Class)

You are developing a simple student information management module.

Prompt:



Code:



```
▶ class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")

# Create a sample student object
student1 = Student("Alice Smith", "CS101", "Computer Science")

# Display the student's details
student1.display_details()
```

Output:

```
*** Student Name: Alice Smith
    Roll Number: CS101
    Branch: Computer Science
```

Explanation:

- A class named Student is created to represent student information.
- The `__init__()` constructor is defined and is called automatically when a Student object is created.
- The constructor initializes the student's name, roll number, and branch.
- A method `display_details()` is defined to print the student details.
- A Student object is created by passing values for name, roll number, and branch.
- The `display_details()` method is called using the object.
- The student information is displayed on the console.

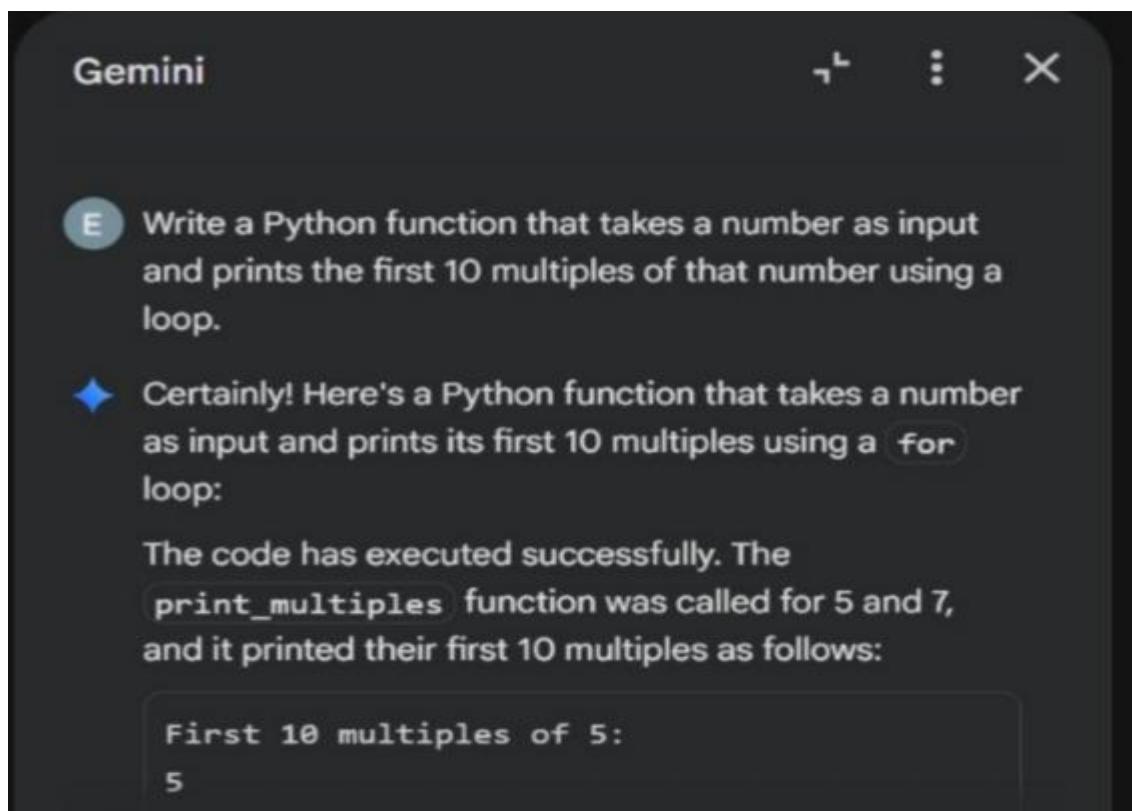
Task-02:

Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number.

Prompt:



Code:

```
def print_multiples(number):  
    print(f"First 10 multiples of {number}:")  
    for i in range(1, 11): # Loop from 1 to 10 (inclusive)  
        multiple = number * i  
        print(multiple)  
  
# Example usage:  
print_multiples(5)  
print_multiples(7)
```

Output:

```
*** First 10 multiples of 5:  
5  
10  
15  
20  
25  
30  
35  
40  
45  
50  
First 10 multiples of 7:  
7  
14  
21  
28  
35  
42  
49  
56  
63  
70
```

Explanation:

- A function is defined that accepts a number as input.
- A loop is used to repeat the process 10 times.
- In each iteration, the input number is multiplied by the loop counter.
- The result of the multiplication gives one multiple of the number.
- Each multiple is printed on the console.
- The loop continues until the first 10 multiples are displayed.
- When the loop ends, the program stops execution.

Task-03:

Conditional Statements (Age Classification)
Scenario

You are building a basic classification system based on age.

Prompt:

E Create a Python function that takes age as input and classifies it into groups such as child, teenager, adult, and senior using nested if-elif-else statements.

◆ Okay, here's a Python function `classify_age` that takes an age as input and categorizes it into 'Child', 'Teenager', 'Adult', or 'Senior' using nested if-elif-else statements. I've also included some example calls to demonstrate its functionality.

[Explain the logic of the 'classify_age' function](#)

[Refactor 'classify_age' to use a single if-elif-else chain](#)

[Apply 'classify_age' to a list of ages](#)

What can I help you build?

+ Gemini 2.5 Flash ▾ ▶

Code:

```
def classify_age(age):
    if age >= 0:
        if age <= 12:
            return "Child"
        elif age <= 19:
            return "Teenager"
        elif age <= 64:
            return "Adult"
        else:
            return "Senior"
    else:
        return "Invalid Age (Age cannot be negative)"

# Test cases
print(f"Age 5 is: {classify_age(5)}")
print(f"Age 15 is: {classify_age(15)}")
print(f"Age 30 is: {classify_age(30)}")
print(f"Age 70 is: {classify_age(70)}")
print(f"Age -2 is: {classify_age(-2)}")
```

Output:

```
Age 5 is: Child
Age 15 is: Teenager
Age 30 is: Adult
Age 70 is: Senior
Age -2 is: Invalid Age (Age cannot be negative)
```

Explanation:

A function is defined that accepts age as an input value.

- The program checks the age using if-elif-else conditions.
- If the age is less than a certain value, it is classified as a child.
- If the age falls in the next range, it is classified as a teenager.
- If the age is in the adult range, it is classified as an adult.
- If the age is above the adult range, it is classified as a senior.

- Only one condition is executed because once a condition is true, the remaining checks are skipped.
- The function returns or prints the appropriate age group.

Task-04:

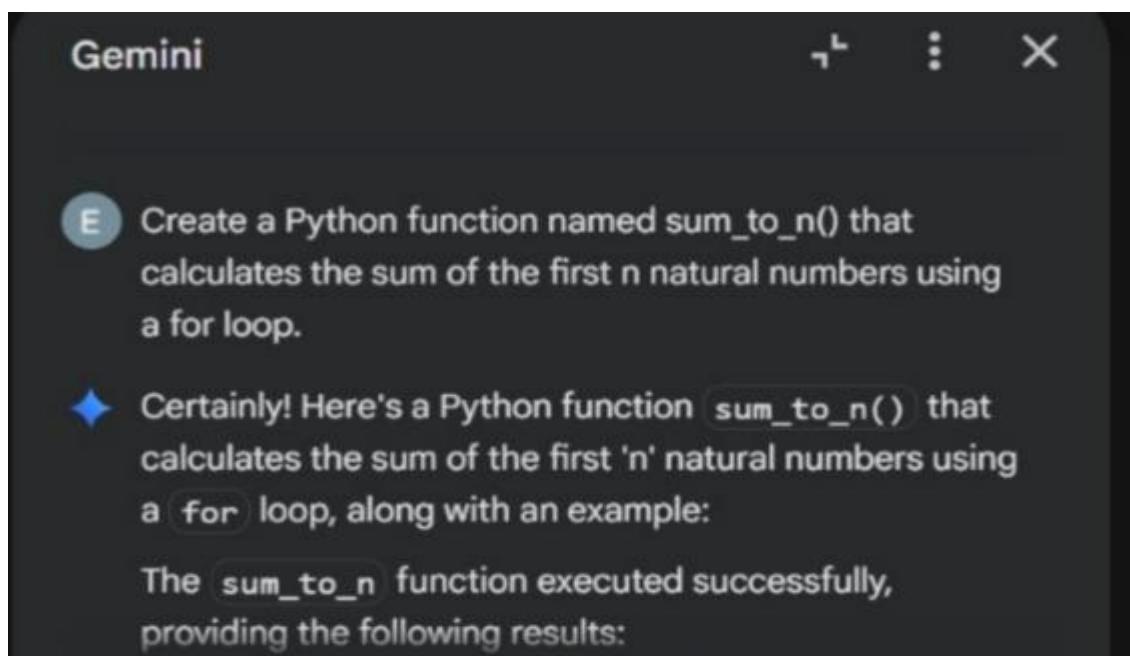
For and While Loops (Sum of First n Numbers)

Scenario

You need to calculate the sum of the first n natural numbers.

a) Using For Loop:

Prompt:



Code:

```
def sum_to_n(n):
    if n < 0:
        return "Input must be a non-negative integer."

    total_sum = 0
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage:
print(f"The sum of the first 5 natural numbers is: {sum_to_n(5)}")
print(f"The sum of the first 10 natural numbers is: {sum_to_n(10)}")
print(f"The sum of the first 0 natural numbers is: {sum_to_n(0)}")
print(f"Trying with a negative number: {sum_to_n(-3)}")
```

Output:

```
The sum of the first 5 natural numbers is: 15
The sum of the first 10 natural numbers is: 55
The sum of the first 0 natural numbers is: 0
Trying with a negative number: Input must be a non-negative integer.
```

b)using While Loop:

Prompt:

E Create a Python function named `sum_to_n()` that calculates the sum of the first `n` natural numbers using a `while` loop.

◆ Certainly! Here's a Python function `sum_to_n()` that calculates the sum of the first '`n`' natural numbers using a `while` loop, along with an example:

⌚ Working...

Code:

```
def sum_to_n_while(n):
    if n < 0:
        return "Input must be a non-negative integer."

    total_sum = 0
    count = 1
    while count <= n:
        total_sum += count
        count += 1
    return total_sum

# Example usage:
print(f"The sum of the first 5 natural numbers using while loop is: {sum_to_n_while(5)}")
print(f"The sum of the first 10 natural numbers using while loop is: {sum_to_n_while(10)}")
print(f"The sum of the first 0 natural numbers using while loop is: {sum_to_n_while(0)}")
print(f"Trying with a negative number: {sum_to_n_while(-3)})")
```

Output:

```
The sum of the first 5 natural numbers using while loop is: 15
The sum of the first 10 natural numbers using while loop is: 55
The sum of the first 0 natural numbers using while loop is: 0
Trying with a negative number: Input must be a non-negative integer.
```

Explanation:

- A function named `sum_to_n()` is defined to calculate the sum of natural numbers.
- The function takes an integer `n` as input.
- A variable (for example, `total`) is initialized to 0 to store the sum.
- A for loop runs from 1 to `n`.
- In each iteration, the current number is added to `total`.

- After the loop finishes, total contains the sum of the first n natural numbers.
- The function returns or prints the final sum.
- The output is displayed for the given sample input.

Task-05:

Classes (Bank Account Class)

Scenario

You are designing a basic banking application.

Prompt:

E Create a Python class named BankAccount with attributes for account holder name and balance. Add methods deposit(), withdraw(), and check_balance(). Show deposit and withdrawal operations with updated balance. Include comments in the code and explain how the class and methods work.

Code:

```

class BankAccount:
    def __init__(self, account_holder_name, initial_balance=0):
        self.account_holder_name = account_holder_name
        self.initial_balance = initial_balance
        self.balance = initial_balance
        if initial_balance < 0:
            print("Initial balance cannot be negative. Setting balance to 0.")
            self.balance = 0

    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"Deposit of ${amount:.2f} successful!")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """Withdraws a specified amount from the account.

        Args:
            amount (float): The amount to withdraw.
        """
        if amount < 0:
            print("Withdrawal amount must be positive.")
        elif self.balance < amount:
            print(f"Insufficient funds. Current balance: ${self.balance:.2f}")
        else:
            self.balance -= amount
            print(f"Withdrawal of ${amount:.2f} successful!")

    def check_balance(self):
        """Prints the current balance of the account.

        Returns:
            str: Account Holder: [self.account_holder_name], Current Balance: ${self.balance:.2f}"
        """
        print(f"--- Demonstrating the BankAccount class ---")
        print(f"1. --- Creating Account ---")
        my_account = BankAccount("John Doe", 1000.00)
        print(f"2. --- Perform a deposit operation ---")
        my_account.deposit(500.50)
        print(f"3. --- Perform a withdrawal operation ---")
        my_account.withdraw(200.75)
        print(f"4. Attempt to withdraw more than available funds")
        my_account.withdraw(1000.00)
        print(f"5. Attempt invalid operations (negative deposit/withdrawal)")
        my_account.deposit(-100)
        my_account.withdraw(5)

    def __str__(self):
        return f"Account Holder: {self.account_holder_name}, Current Balance: ${self.balance:.2f}"

```

Output:

```

my_account.withdraw(0)
...
--- Creating Account ---
Account Holder: John Doe, Current Balance: $1000.00
--- Deposit Operation ---
Deposit of $500.50 successful.
Account Holder: John Doe, Current Balance: $1500.50

--- Withdrawal Operation ---
Withdrawal of $200.75 successful.
Account Holder: John Doe, Current Balance: $1299.75

--- Attempting Over-withdrawal ---
Insufficient funds.
Account Holder: John Doe, Current Balance: $1299.75

--- Attempting Invalid Operations ---
Deposit amount must be positive.
Account Holder: John Doe, Current Balance: $1299.75
Withdrawal amount must be positive.
Account Holder: John Doe, Current Balance: $1299.75

```

Explanation:

- A class named `BankAccount` is created to represent a bank account.
- The constructor (`__init__`) initializes the account holder's name and the starting balance.

- The deposit() method is used to add a given amount to the current balance.
- The withdraw() method subtracts an amount from the balance after checking if sufficient funds are available.
- If the balance is not enough, the withdrawal is not allowed and a message is shown.
- The check_balance() method displays the current account balance.
- An object of the BankAccount class is created to perform banking operations.
- Deposit and withdrawal methods are called, and the updated balance is shown after each operation.