
LAB ASSIGNMENT – 6.3

Specialization : AIML
Name of Student : D.LAXMAN
Enrollment-No : 2403A51283
Batch No : 01
Date : 10-09-2025

Task Description:

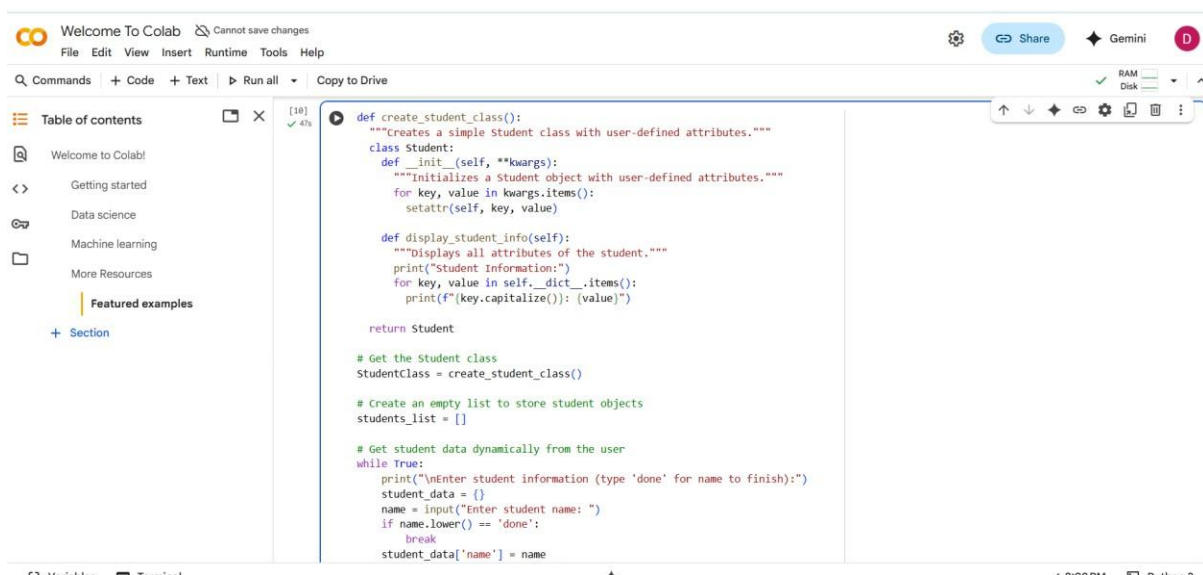
- Use AI to complete a Student class with attributes and a method.
- Check output
- Analyze the code generated by AI tool

Prompt:

create a python function that completes a Student class with attributes and a display() method where attributes of object are user

defined. Make sure it should accept multiple student records which are user defined

Code:



The screenshot shows the Google Colab interface. The left sidebar contains a 'Table of contents' and a 'Welcome to Colab!' message. The main code editor displays the following Python code:

```
[10] ✓ 47s
def create_student_class():
    """Creates a simple Student class with user-defined attributes."""
    class Student:
        def __init__(self, **kwargs):
            """Initializes a Student object with user-defined attributes."""
            for key, value in kwargs.items():
                setattr(self, key, value)


        def display_student_info(self):
            """Displays all attributes of the student."""
            print("Student Information:")
            for key, value in self.__dict__.items():
                print(f"{key.capitalize()}: {value}")

    return Student

# Get the Student class
StudentClass = create_student_class()

# Create an empty list to store student objects
students_list = []

# Get student data dynamically from the user
while True:
    print("\nEnter student information (type 'done' for name to finish):")
    student_data = {}
    name = input("Enter student name: ")
    if name.lower() == 'done':
        break
    student_data['name'] = name
```



The screenshot shows the Google Colab interface with the second part of the code in the main editor:

```
# Get student data dynamically from the user
while True:
    print("\nEnter student information (type 'done' for name to finish):")
    student_data = {}
    name = input("Enter student name: ")
    if name.lower() == 'done':
        break
    student_data['name'] = name

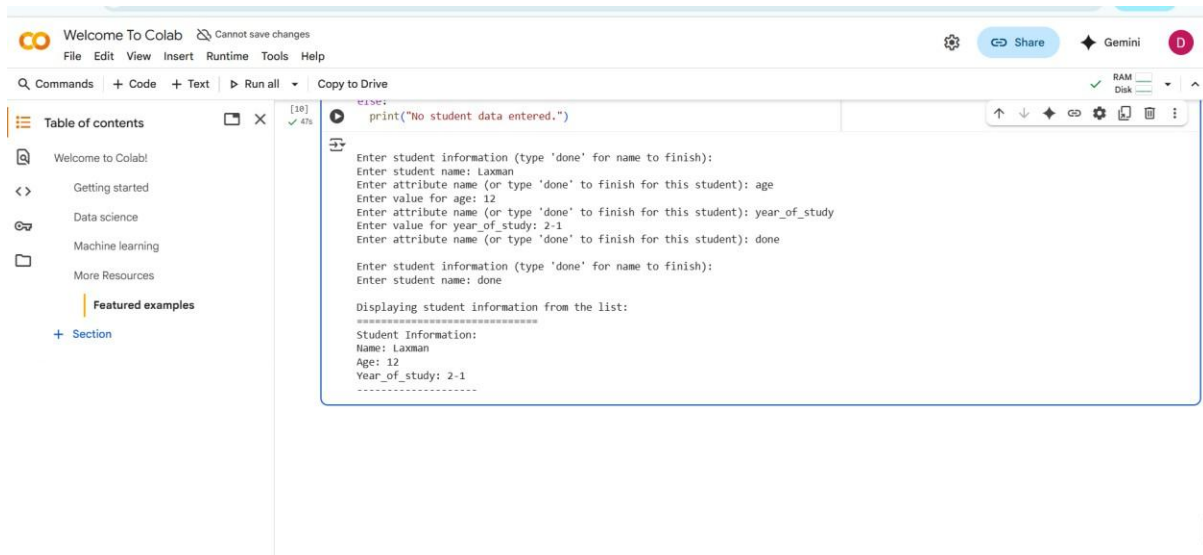
    while True:
        attribute_name = input("Enter attribute name (or type 'done' to finish for this student): ")
        if attribute_name.lower() == 'done':
            break
        attribute_value = input(f"Enter value for {attribute_name}: ")
        student_data[attribute_name] = attribute_value

    if student_data: # Only create student if some data was entered
        new_student = StudentClass(**student_data)
        students_list.append(new_student)

# Now you can access and display information for students in the list
print("\nDisplaying student information from the list:")
print("-" * 30)
if students_list:
    for student in students_list:
        student.display_student_info()
        print("-" * 20) # Separator for clarity
else:
    print("No student data entered.")
```

Output:

THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, a 'Welcome To Colab' message, and a 'Cannot save changes' warning. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a 'Table of contents' with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', and 'More Resources'. The main area displays a Python code cell with the following code:

```
print("No student data entered.")

Enter student information (type 'done' for name to finish):
Enter student name: Laxman
Enter attribute name (or type 'done' to finish for this student): age
Enter value for age: 12
Enter attribute name (or type 'done' to finish for this student): year_of_study
Enter value for year_of_study: 2-1
Enter attribute name (or type 'done' to finish for this student): done

Enter student information (type 'done' for name to finish):
Enter student name: done

Displaying student information from the list:
=====
Student Information:
Name: Laxman
Age: 12
Year_of_study: 2-1
=====
```

Explanation:

- `create_student_class()` function: This function defines and returns the Student class.

- Student Class: This class represents a student.
- `__init__(self, **kwargs)`: This is the constructor. It's designed to accept any number of keyword arguments (`**kwargs`) when you create a Student object. This allows you to define the student's attributes (like name, age, major, etc.) dynamically.
- `display_student_info(self)`: This method iterates through all the attributes that were set during initialization and prints them in a formatted way.
- Dynamic Input and List Storage: The code after the function definition includes a loop that prompts the user to enter student information (name and other attributes) during runtime. Each set of entered data is used to create

a Student object, and these objects are stored in the students_list. Finally, it iterates through the students_list and displays the information for each student.

Observation:

THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.

THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.

Description Task2:

- 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:

Create a Python code that first 10 multiples of a user defined number using nested loop concept

Code:



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, a 'Welcome To Colab' message, and a 'Cannot save changes' warning. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a 'Table of contents' with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', and 'More Resources'. The main area displays a Python code cell with the following content:

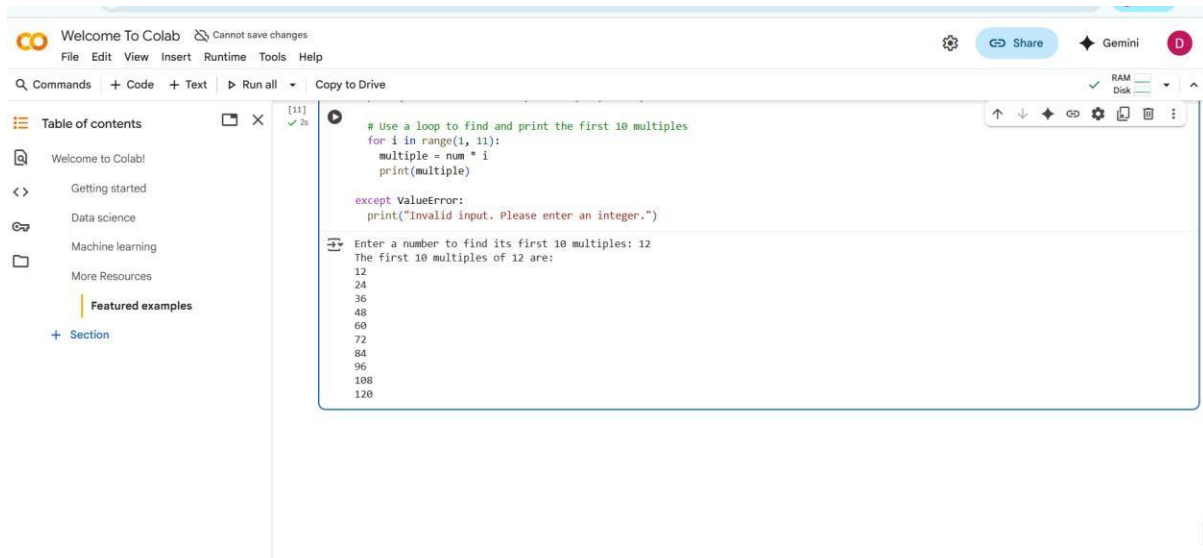
```
=====
Student Information:
Name: Laxman
Age: 12
Year_of_study: 2-1
=====

[11] ✓ 2s • Get the number from the user
try:
    num = int(input("Enter a number to find its first 10 multiples: "))

    print(f"The first 10 multiples of {num} are:")

    # Use a loop to find and print the first 10 multiples
    for i in range(1, 11):
        multiple = num * i
        print(multiple)

except ValueError:
    print("Invalid input. Please enter an integer.")
```



The screenshot shows the Google Colab interface. The top bar includes the Colab logo, 'Welcome To Colab', and a 'Cannot save changes' warning. The menu bar has 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a 'Table of contents' with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', 'More Resources', and 'Featured examples'. The main code editor area contains a Python script that uses a for loop to find and print the first 10 multiples of a user-entered number. The output shows the first 10 multiples of 12.

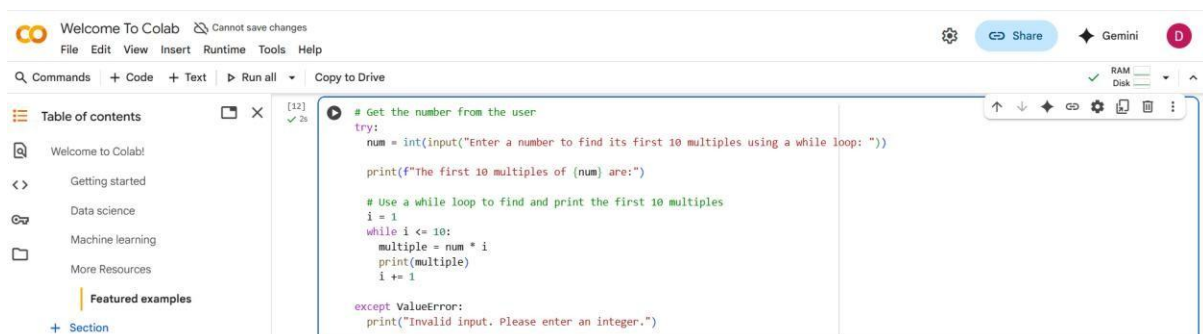
```
# Use a loop to find and print the first 10 multiples
for i in range(1, 11):
    multiple = num * i
    print(multiple)

except ValueError:
    print("Invalid input. Please enter an integer.")
```

Enter a number to find its first 10 multiples: 12
The first 10 multiples of 12 are:
12
24
36
48
60
72
84
96
108
120

Ask AI to generate code using other controlled looping

Code:



The screenshot shows the Google Colab interface with a Python script that uses a while loop to find and print the first 10 multiples of a user-entered number. The code includes a try-except block to handle invalid input.

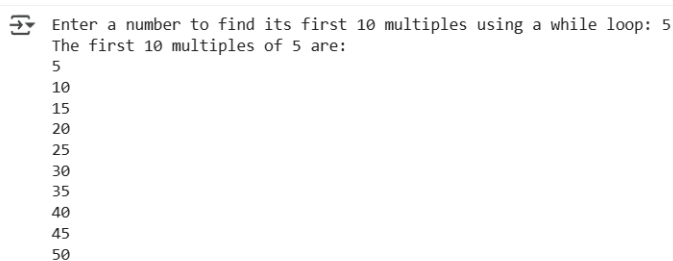
```
# Get the number from the user
try:
    num = int(input("Enter a number to find its first 10 multiples using a while loop: "))

    print(f"The first 10 multiples of {num} are:")

    # Use a while loop to find and print the first 10 multiples
    i = 1
    while i <= 10:
        multiple = num * i
        print(multiple)
        i += 1

except ValueError:
    print("Invalid input. Please enter an integer.")
```

Output:



The screenshot shows the output of the while loop code. It prompts the user to enter a number, and the output displays the first 10 multiples of 5.

```
Enter a number to find its first 10 multiples using a while loop: 5
The first 10 multiples of 5 are:
5
10
15
20
25
30
35
40
45
50
```

Explanation:

I understand above generated codes

Observation:

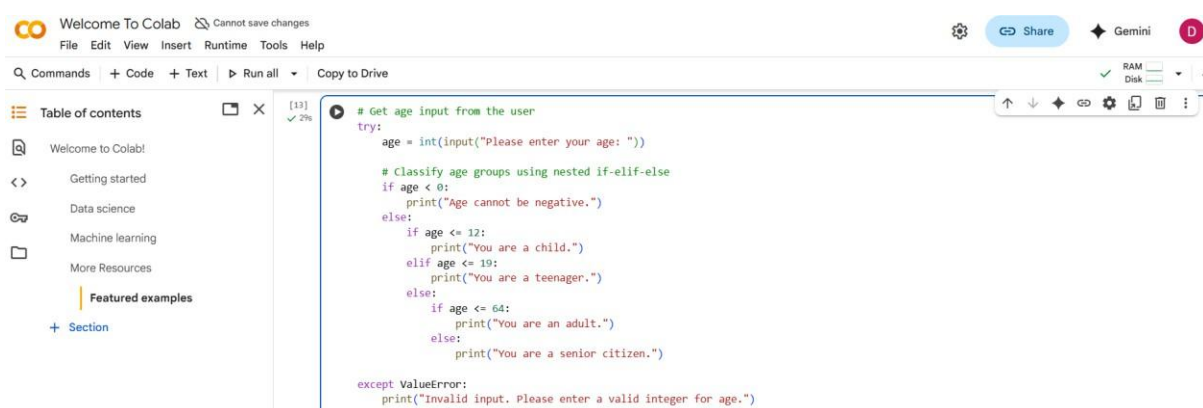
THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.

Description Task3:

- 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

Prompt:

create a python code which writes nested if-elif-else conditionals to classify age groups.



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, 'Welcome To Colab', a warning 'Cannot save changes', and a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. On the right, there are buttons for 'Share', 'Gemini', and a user profile icon. Below the top bar, a search bar and tabs for 'Commands', '+ Code', '+ Text', 'Run all', and 'Copy to Drive' are visible. The left sidebar contains a 'Table of contents' with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', 'More Resources', and 'Featured examples'. The main code editor area displays the following Python code:

```
# Get age input from the user
try:
    age = int(input("Please enter your age: "))

    # Classify age groups using nested if-elif-else
    if age < 0:
        print("Age cannot be negative.")
    else:
        if age <= 12:
            print("You are a child.")
        elif age <= 19:
            print("You are a teenager.")
        else:
            if age <= 64:
                print("You are an adult.")
            else:
                print("You are a senior citizen.")

except ValueError:
    print("Invalid input. Please enter a valid integer for age.")
```

Output:

```
except ValueError:
    print("Invalid input. Please enter a valid integer for age.")

Please enter your age: 18
You are a teenager.
```

Ask AI to generate code using other conditional statements

```
[15] ✓ 1s Get age input from the user
try:
    age = int(input("Please enter your age: "))

    # Classify age groups using a flatter if-elif-elif-else structure
    if age < 0:
        print("Age cannot be negative.")
    elif age <= 12:
        print("You are a child.")
    elif age <= 19:
        print("You are a teenager.")
    elif age <= 64:
        print("You are an adult.")
    else:
        print("You are a senior citizen.")

except ValueError:
    print("Invalid input. Please enter a valid integer for age.")

Please enter your age: 11
You are a child.
```

Explanation:

I Understand the above generated Codes

Observation:

THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.

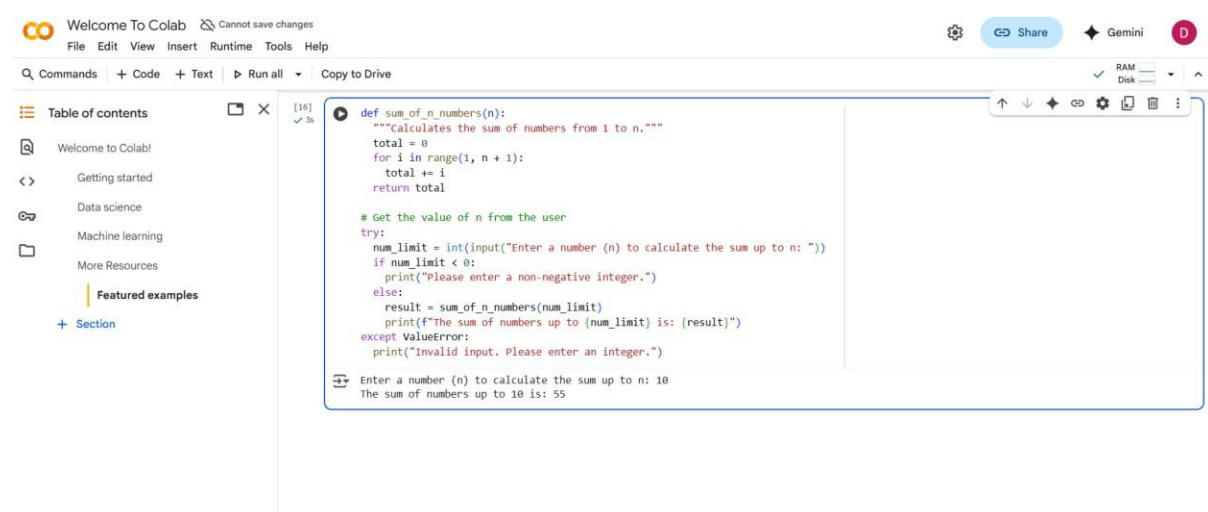
Task Description:

- 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:

Create a Python Code that calculates the sum of `n` terms where `n` must be user defined

Code & Output:



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, 'Welcome To Colab', and a 'Cannot save changes' warning. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a 'Table of contents' with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', and 'More Resources'. The main area displays a Python code cell with the following code:

```
[16] ✓ 3s def sum_of_n_numbers(n):  
    """Calculates the sum of numbers from 1 to n."""  
    total = 0  
    for i in range(1, n + 1):  
        total += i  
    return total  
  
# Get the value of n from the user  
try:  
    num_limit = int(input("Enter a number (n) to calculate the sum up to n: "))  
    if num_limit < 0:  
        print("Please enter a non-negative integer.")  
    else:  
        result = sum_of_n_numbers(num_limit)  
        print(f"The sum of numbers up to {num_limit} is: {result}")  
except ValueError:  
    print("Invalid input. Please enter an integer.")
```

The output of the code cell is:

```
Enter a number (n) to calculate the sum up to n: 10  
The sum of numbers up to 10 is: 55
```

Observation:

THERE IS NOTHING TO CORRECT. THE CODE IS EXECUTED PERFECTLY BY USING THE FOR AND GENERATED THE MINIMAL CODE WHICH MEETS MY REQUIREMENT AND SIMPLE OUTPUT.

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:

create a class with methods
deposit(),withdrawa(),balance and display()

Code:



```
class BankAccount:
    """A simple bank account class."""

    def __init__(self, account_holder, initial_balance=0):
        """Initializes a new bank account."""
        self.account_holder = account_holder
        self.balance = initial_balance
        print(f"Account created for {self.account_holder} with an initial balance of ${self.balance:.2f}")

    def deposit(self, amount):
        """Deposits money into the account."""
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """Withdraws money from the account."""
        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("Withdrawal amount must be positive.")

    def display_balance(self):
        """Displays the current account balance."""
```



```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
        else:
            print("Insufficient funds.")

    def display_balance(self):
        print(f"Current balance for {self.account_holder}: ${self.balance:.2f}")

    def display_account_info(self):
        print("\nAccount Information:")
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ${self.balance:.2f}")

# Example usage:
# Create an account
my_account = BankAccount("Alice Smith", 1000)

# Perform some transactions
my_account.deposit(500)
my_account.withdraw(200)
my_account.display_balance()
my_account.withdraw(2000) # Attempt to withdraw more than balance

# Display full account information
my_account.display_account_info()
```

Output:

```
Account created for Alice Smith with an initial balance of $1000.00
Deposited $500.00. New balance: $1500.00
Withdrew $200.00. New balance: $1300.00
Current balance for Alice Smith: $1300.00
Insufficient funds.

Account Information:
Account Holder: Alice Smith
Current Balance: $1300.00
```

Explanation of Code:

- **BankAccount Class:** This class is a blueprint for creating bank account objects.
- **__init__(self, account_holder, initial_balance=0):** This is the constructor method. It's called when you create a

new BankAccount object. It sets the account_holder name and an optional initial_balance (defaulting to 0).

- deposit(self, amount): This method allows you to add money to the account. It checks if the deposit amount is positive and updates the balance.
- withdraw(self, amount): This method allows you to take money out of the account. It checks if the withdrawal amount is positive and if there are sufficient funds before updating the balance.
- display_balance(self): This method simply prints the current balance of the account.
- display_account_info(self): This method prints both the account holder's name and the current balance.

Observation:

**THERE IS NOTHING TO CORRECT. THE CODE
IS EXECUTED PERFECTLY BY USING THE FOR
AND GENERATED THE MINIMAL CODE
WHICH MEETS MY REQUIREMENT AND
SIMPLE OUTPUT.**