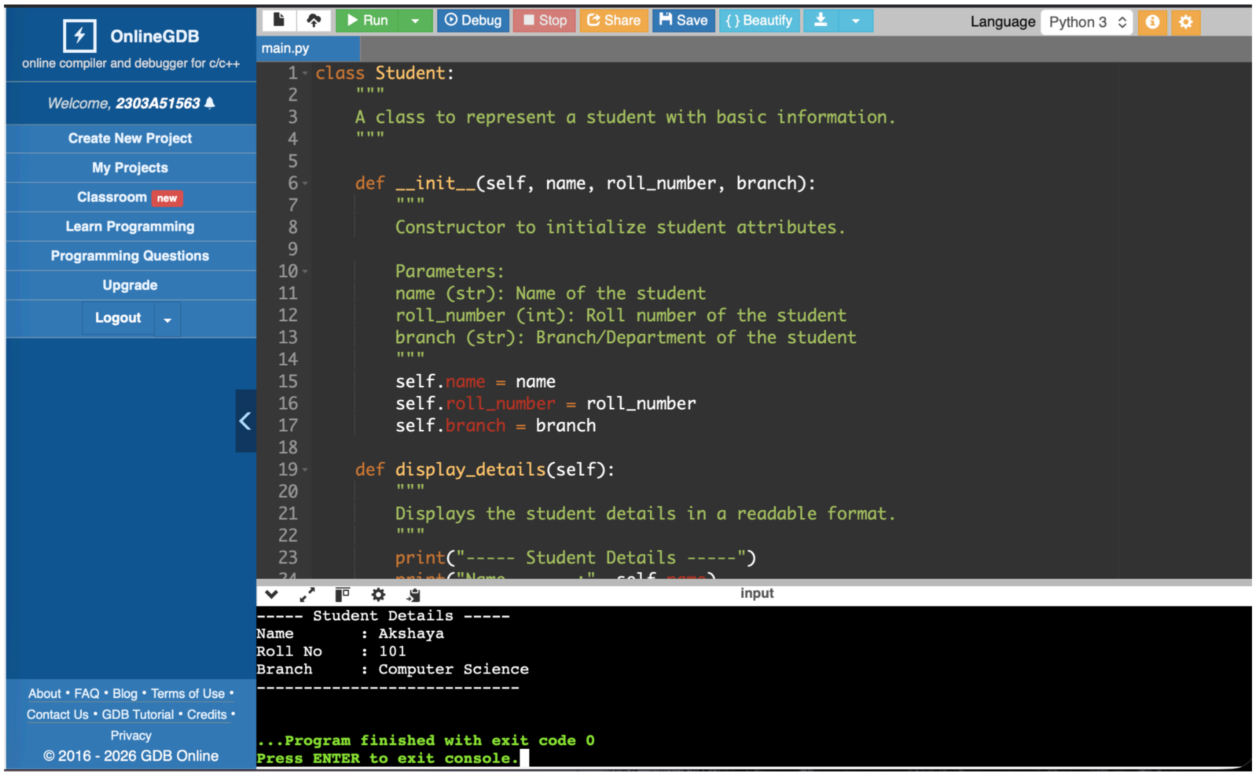
2303A51563
batch=10
Task Description #1: Classes (Student Class)
Scenario
You are developing a simple student information management module.
Task
• Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.
• The class should include attributes such as name, roll number, and branch.
• Add a method display_details() to print student information.
• Execute the code and verify the output.
• Analyze the code generated by the AI tool for correctness and clarity.
Expected Output #1
• A Python class with a constructor (__init__) and a display_details() method.
• Sample object creation and output displayed on the console.
• Brief analysis of AI-generated code.
Code
```
class Student:
    """
    A class to represent a student with basic information.
    """

    def __init__(self, name, roll_number, branch):
        """
        Constructor to initialize student attributes.

        Parameters:
        name (str): Name of the student
        roll_number (int): Roll number of the student
        branch (str): Branch/Department of the student
        """
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        """
        Displays the student details in a readable format.
        """
        print("----- Student Details -----")
        print("Name       :", self.name)
        print("Roll No    :", self.roll_number)
        print("Branch     :", self.branch)
        print("---------------------------")
```

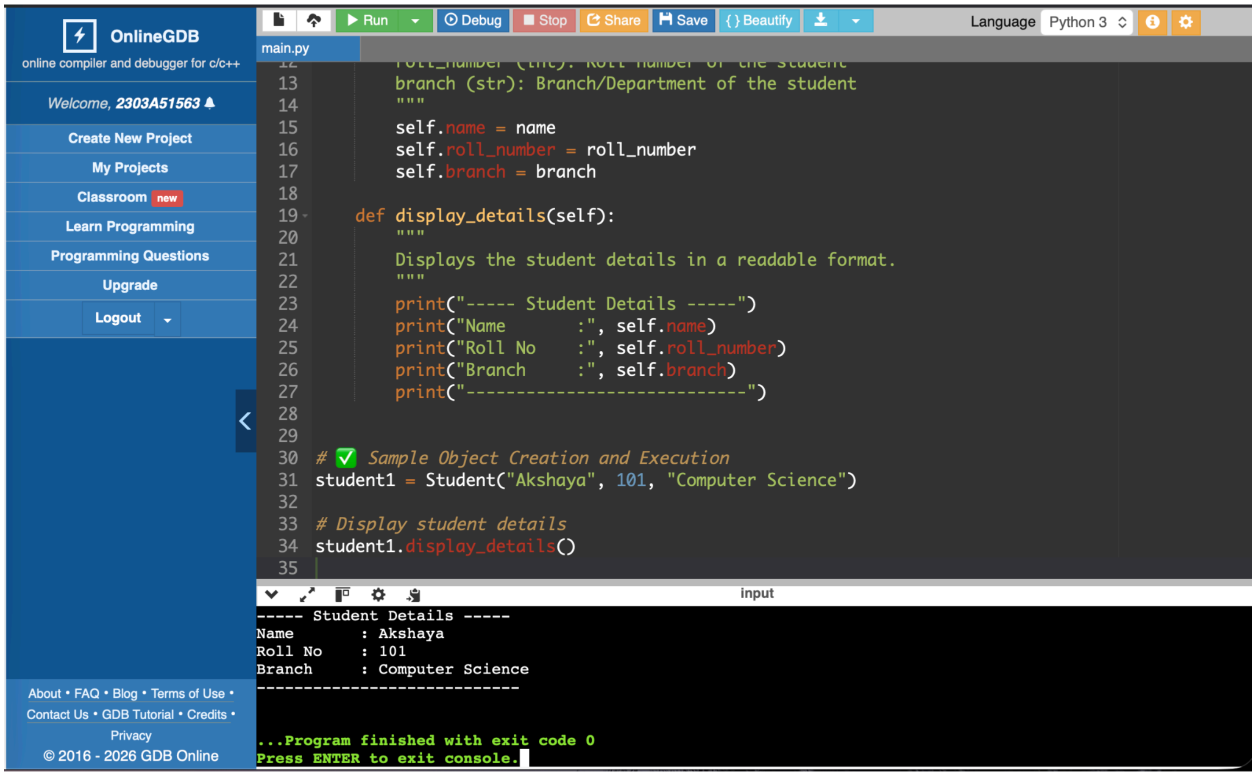
# ✅ Sample Object Creation and Execution
student1 = Student("Akshaya", 101, "Computer Science")

# Display student details
student1.display_details()



```
class Student:
    """
    A class to represent a student with basic information.
    """

    def __init__(self, name, roll_number, branch):
        """
        Constructor to initialize student attributes.

        Parameters:
        name (str): Name of the student
        roll_number (int): Roll number of the student
        branch (str): Branch/Department of the student
        """
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        """
        Displays the student details in a readable format.
        """
        print("----- Student Details -----")
```

```
----- Student Details -----
Name       : Akshaya
Roll No    : 101
Branch     : Computer Science
----------------------------


...Program finished with exit code 0
Press ENTER to exit console.
```



```python
        roll_number (int): Roll number of the student
        branch (str): Branch/Department of the student
        """
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        """
        Displays the student details in a readable format.
        """
        print("----- Student Details -----")
        print("Name       :", self.name)
        print("Roll No    :", self.roll_number)
        print("Branch     :", self.branch)
        print("---------------------------")


# ✅ Sample Object Creation and Execution
student1 = Student("Akshaya", 101, "Computer Science")

# Display student details
student1.display_details()
```

```
----- Student Details -----
Name       : Akshaya
Roll No    : 101
Branch     : Computer Science
---------------------------


...Program finished with exit code 0
Press ENTER to exit console.
```

ask Description #2: Loops (Multiples of a Number)

Scenario

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

Task

• Prompt the AI tool to generate a function that prints the first 10 multiples of a given number using a loop.

• Analyze the generated loop logic.

• Ask the AI to generate the same functionality using another controlled looping structure (e.g., while instead of for).

Expected Output #2

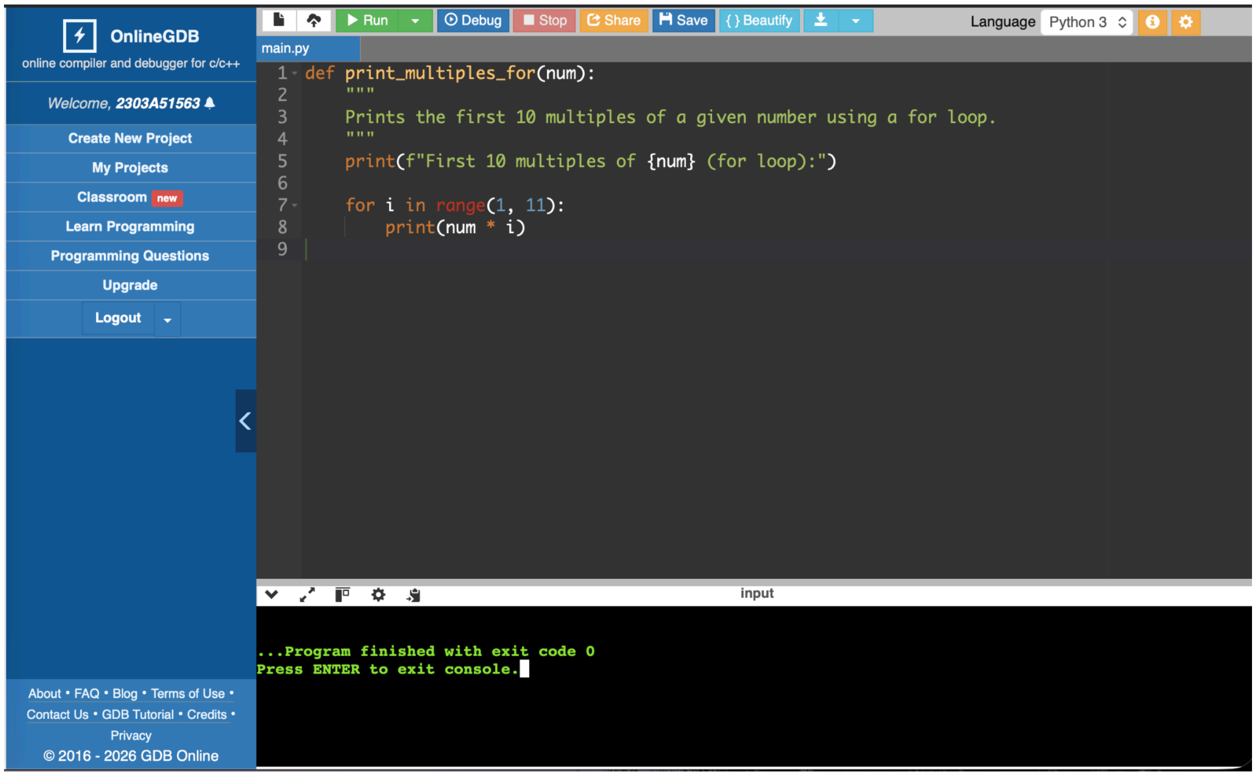
• Correct loop-based Python implementation.

• Output showing the first 10 multiples of a number.

• Comparison and analysis of different looping approaches.

Code

```python
def print_multiples_for(num):
    """
    Prints the first 10 multiples of a given number using a for loop.
    """
    print(f"First 10 multiples of {num} (for loop):")

    for i in range(1, 11):
        print(num * i)
```

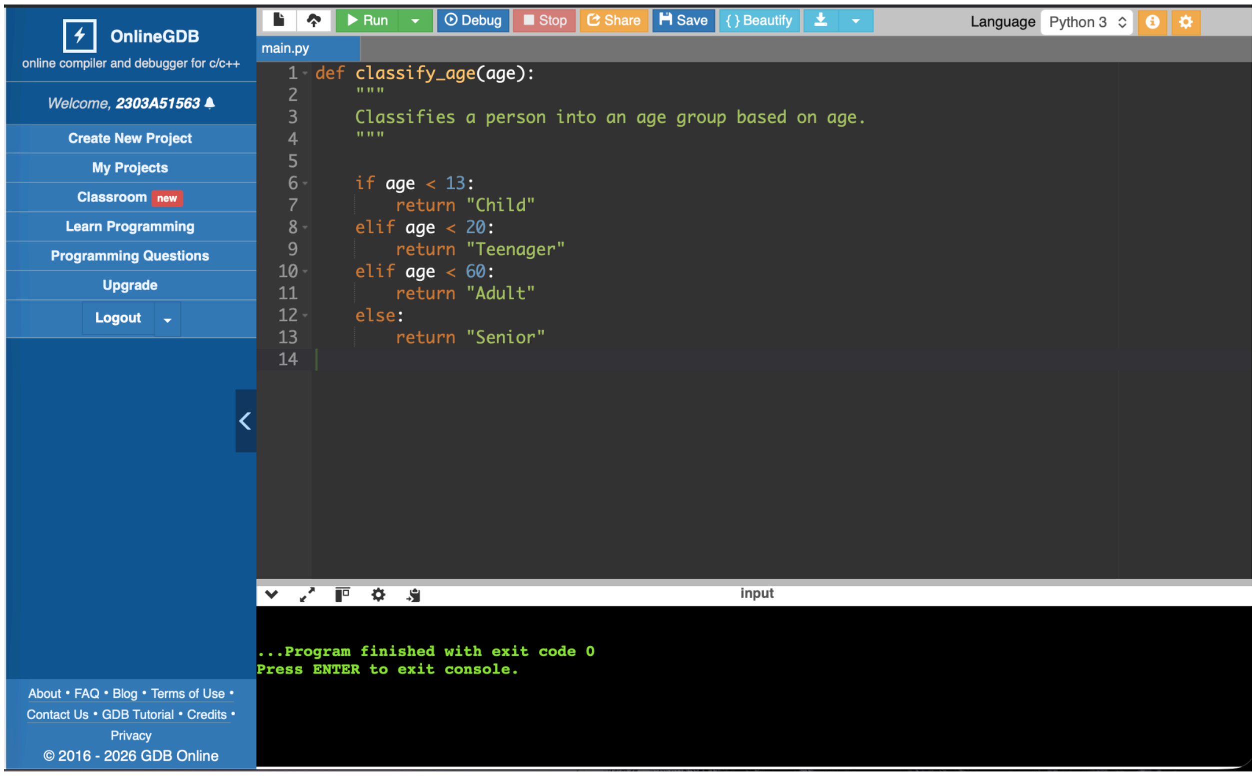


```python
def print_multiples_for(num):
    """
    Prints the first 10 multiples of a given number using a for loop.
    """
    print(f"First 10 multiples of {num} (for loop):")

    for i in range(1, 11):
        print(num * i)

```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

ask Description #3: Conditional Statements (Age Classification)
Scenario
You are building a basic classification system based on age.
Task
• Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).
• Analyze the generated conditions and logic.
• Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).
Expected Output #3
• A Python function that classifies age into appropriate groups.
• Clear and correct conditional logic.
• Explanation of how the conditions work.
Code

```python
def classify_age(age):
    """
    Classifies a person into an age group based on age.
    """

    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"
```



```python
def classify_age(age):
    """
    Classifies a person into an age group based on age.
    """

    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

ask Description #3: Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

Task

• Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).

• Analyze the generated conditions and logic.

• Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).

Expected Output #3

• A Python function that classifies age into appropriate groups.

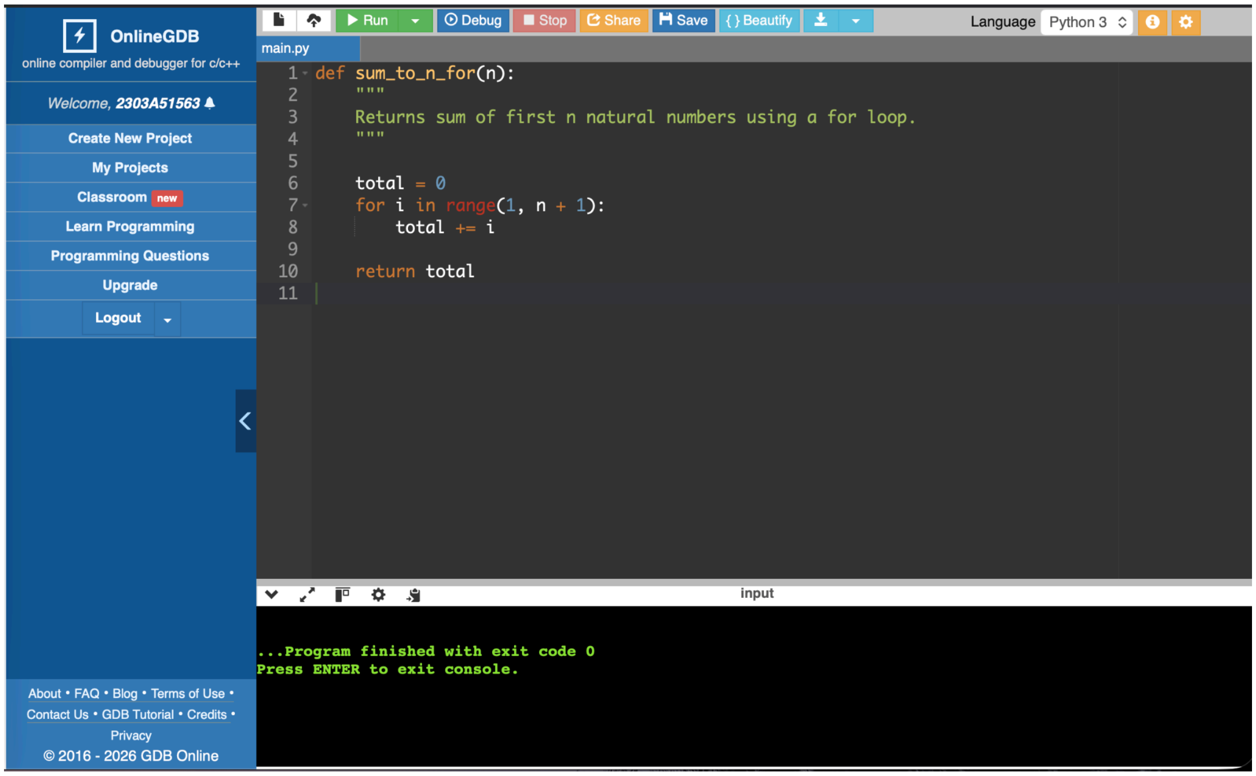
• Clear and correct conditional logic.

• Explanation of how the conditions work.

Code

```
def sum_to_n_for(n):
    """
    Returns sum of first n natural numbers using a for loop.
    """

    total = 0
```

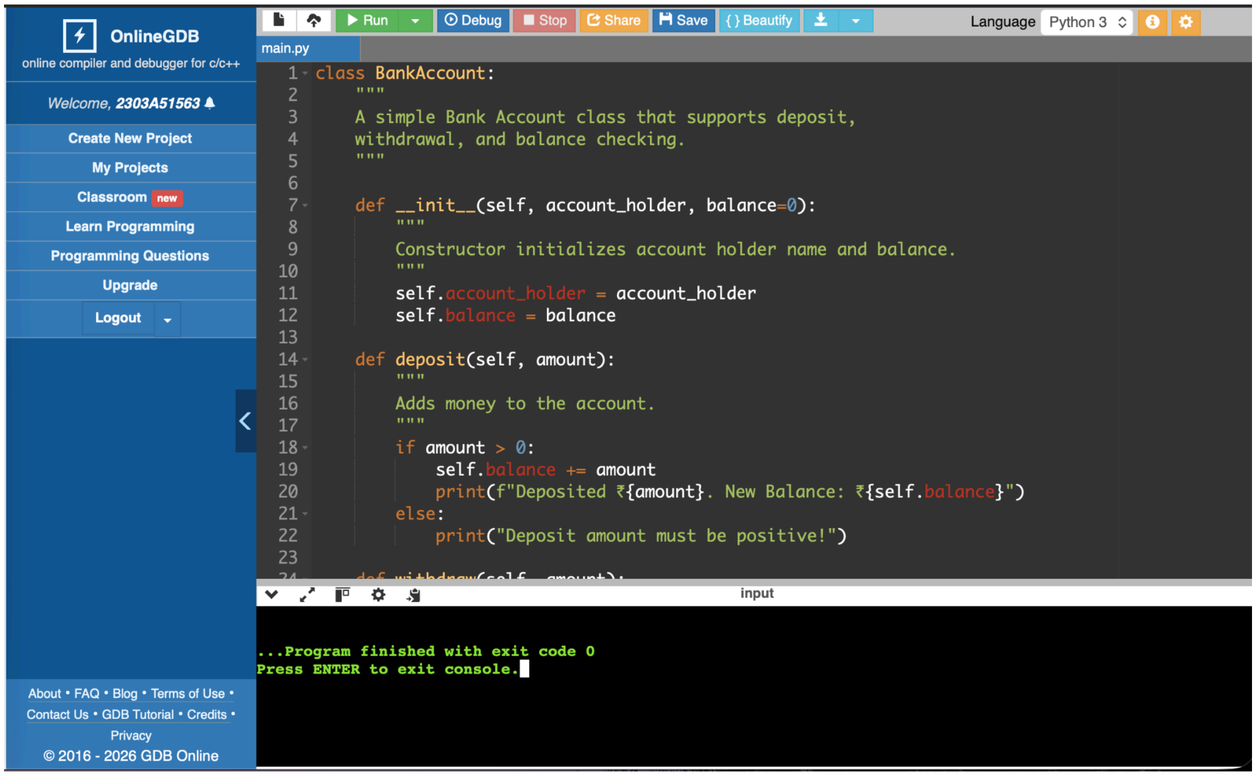
```
for i in range(1, n + 1):
    total += i

return total
```



```python
def sum_to_n_for(n):
    """
    Returns sum of first n natural numbers using a for loop.
    """

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

    return total
```

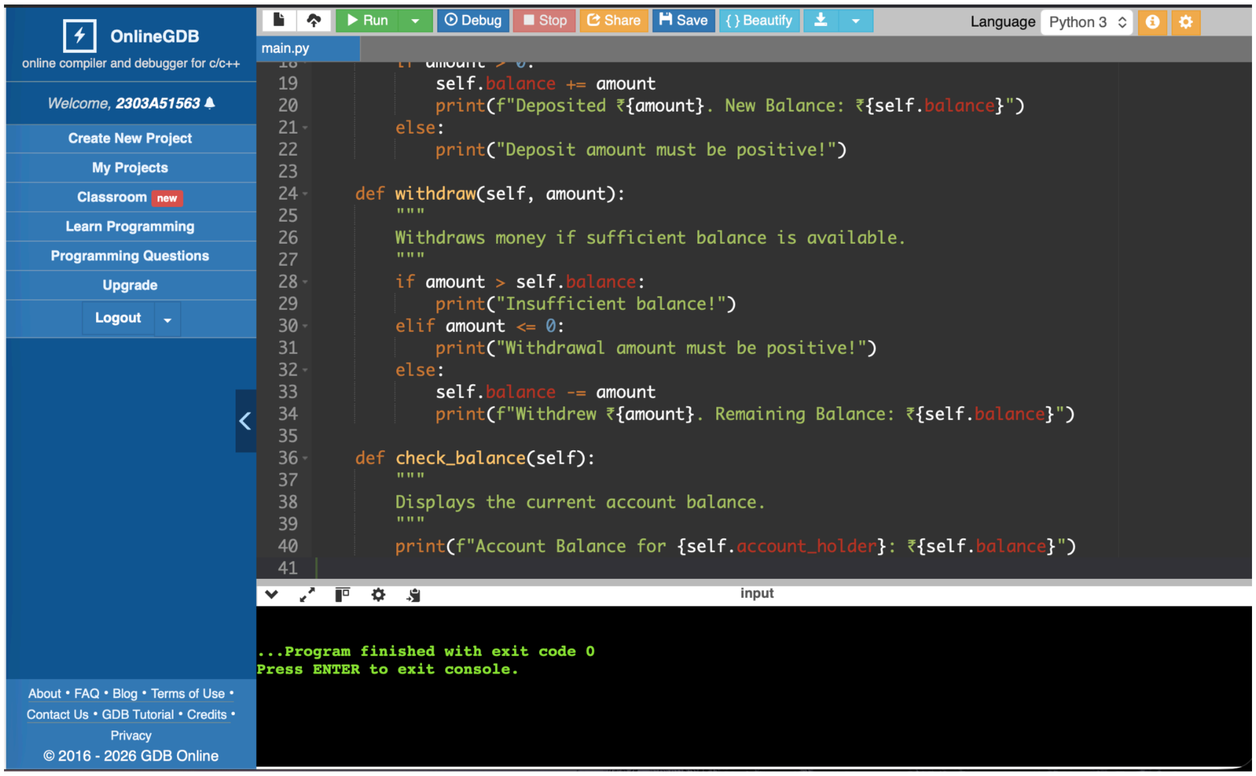
```
...Program finished with exit code 0
Press ENTER to exit console.
```

Task Description #5: Classes (Bank Account Class)
Scenario
You are designing a basic banking application.
Task
• Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().
• Analyze the AI-generated class structure and logic.
• Add meaningful comments and explain the working of the code.
Expected Output #5
• Complete Python Bank Account class.
• Demonstration of deposit and withdrawal operations with updated balance.
• Well-commented code with a clear explanatio



```python
class BankAccount:
    """
    A simple Bank Account class that supports deposit,
    withdrawal, and balance checking.
    """

    def __init__(self, account_holder, balance=0):
        """
        Constructor initializes account holder name and balance.
        """
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """
        Adds money to the account.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposited ₹{amount}. New Balance: ₹{self.balance}")
        else:
            print("Deposit amount must be positive!")

```

```
...Program finished with exit code 0
Press ENTER to exit console.
```



```python
            self.balance += amount
            print(f"Deposited ₹{amount}. New Balance: ₹{self.balance}")
        else:
            print("Deposit amount must be positive!")

    def withdraw(self, amount):
        """
        Withdraws money if sufficient balance is available.
        """
        if amount > self.balance:
            print("Insufficient balance!")
        elif amount <= 0:
            print("Withdrawal amount must be positive!")
        else:
            self.balance -= amount
            print(f"Withdrew ₹{amount}. Remaining Balance: ₹{self.balance}")

    def check_balance(self):
        """
        Displays the current account balance.
        """
        print(f"Account Balance for {self.account_holder}: ₹{self.balance}")

```

```
...Program finished with exit code 0
Press ENTER to exit console.
```