Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals

Week3–Wednesday

Name:B.Tejaskumar

HallTicketNo:2303A51489

Lab Objectives

- ToexploreAI-poweredauto-completionfeaturesforPythonclasses,loops,and conditionals.
- ToanalyzeAI-suggestedlogicforobject-orientedprogrammingandcontrolstructures.
- Toevaluatecorrectness,readability,andcompletenessofAI-generatedcode.

Lab Outcomes (LOs)

Aftercompletingthislab,studentswillbeableto:

- UseAItoolstogeneratePythonclassesandmethods.
- UnderstandandassessAI-suggestedloopconstructs.
- Generateandevaluateconditionalstatements.
- CriticallyanalyzeAI-assistedcode.

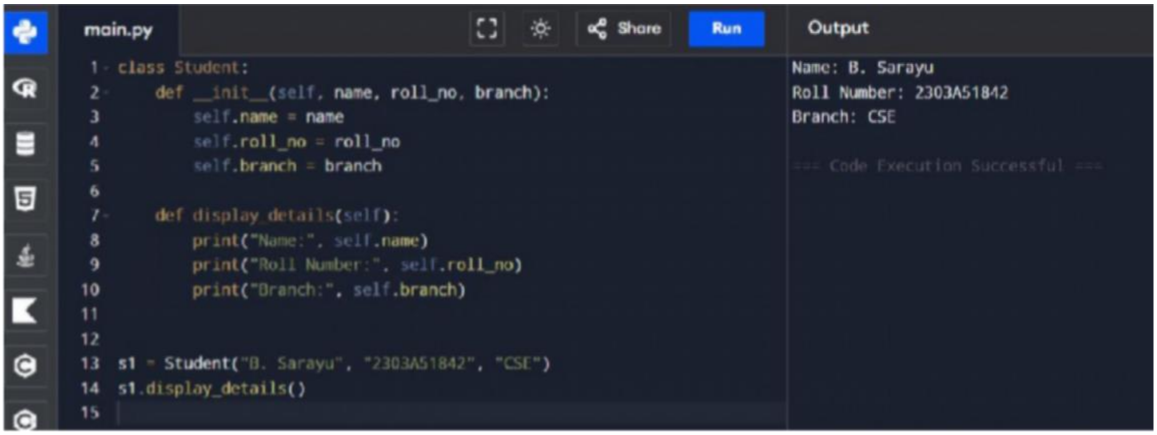
Task 1: Classes – Student Class

PromptUsed:
"GenerateaPythonStudentclasswithname,rollnumber,branch,andamethodtodisplay details."

```
classStudent
  def_init_(self,name,roll_no,branch):
    self.name=name self.roll_no=roll_no
    self.branch=branch

  defdisplay_details(self):
    print("Name:",self.name)
    print("RollNumber:",self.roll_no
    ) print("Branch:",self.branch)



s1=Student("B.t","2303A51489","CSE")
s1.display_details()
```

Output:

```
Name:B.t
```



```python
class Student:
    def __init__(self, name, roll_no, branch):
        self.name = name
        self.roll_no = roll_no
        self.branch = branch

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Branch:", self.branch)


s1 = Student("B. Sarayu", "2303A51842", "CSE")
s1.display_details()
```

```
Name: B. Sarayu
Roll Number: 2303A51842
Branch: CSE

=== Code Execution Successful ===
```

Analysis: TheAI-generatedclassiswell- structured,readable,andcorrectlyusesaconstructorand instancemethod.

Task 2: Loops – Multiples of a Number

PromptUsed "GeneratePythoncodetoprintfirst10multiplesofanumberusingloops."

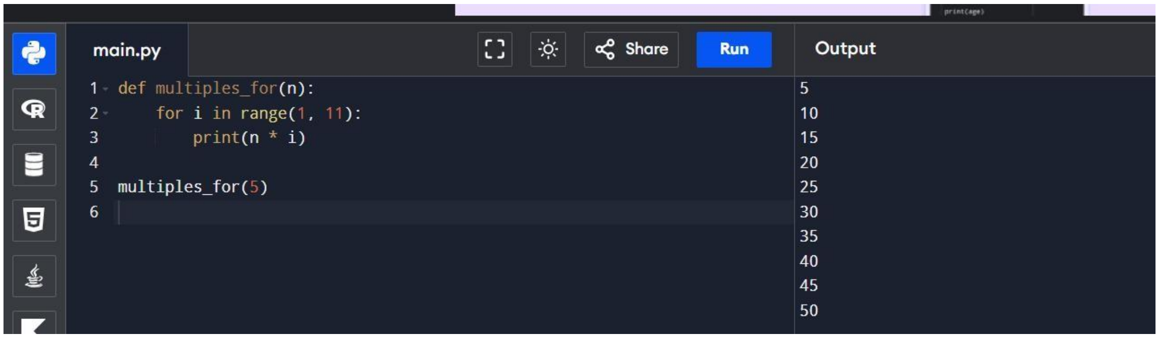
Usingforloop:

```
defmultiples_for(n)
  foriinrange(1,11):
    print(n*i)

multiples_for(5
```

Usingwhileloop:

```
defmultiples_while(n)
  : i=1
  whilei<=10:
    print(n*i)
    i+=1

multiples_while(5
```

Output:
5101520253035404550



```
def multiples_for(n):
    for i in range(1, 11):
        print(n * i)

multiples_for(5)
```

```
5
10
15
20
25
30
35
40
45
50
```

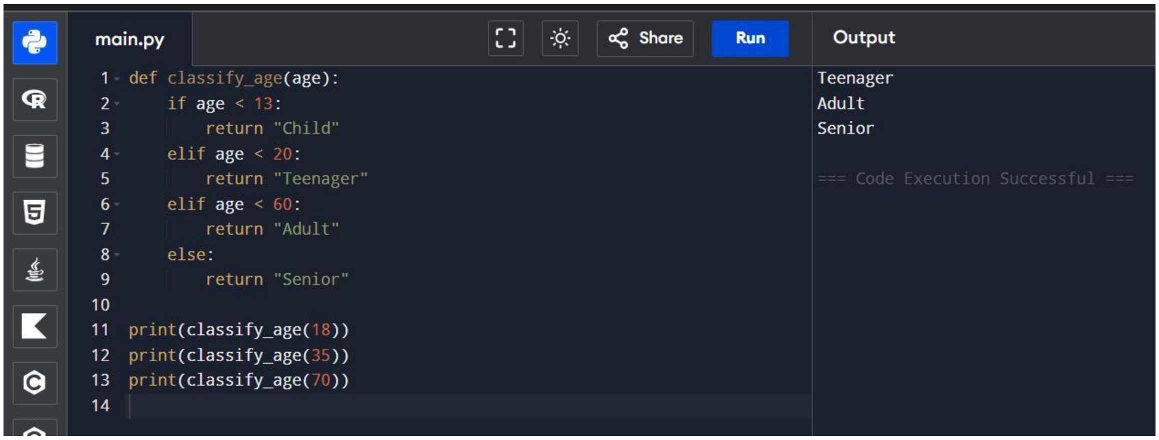
Analysis:
The for loop is concise and readable, while the while loop provides explicit control over iteration.

Task 3: Conditional Statements – Age Classification

Prompt Used:
"Generate Python code to classify age using if-elif-else."

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

print(classify_age(18))
```

Output:
Teenager



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

print(classify_age(18))
print(classify_age(35))
print(classify_age(70))
```

```
Teenager
Adult
Senior

=== Code Execution Successful ===
```

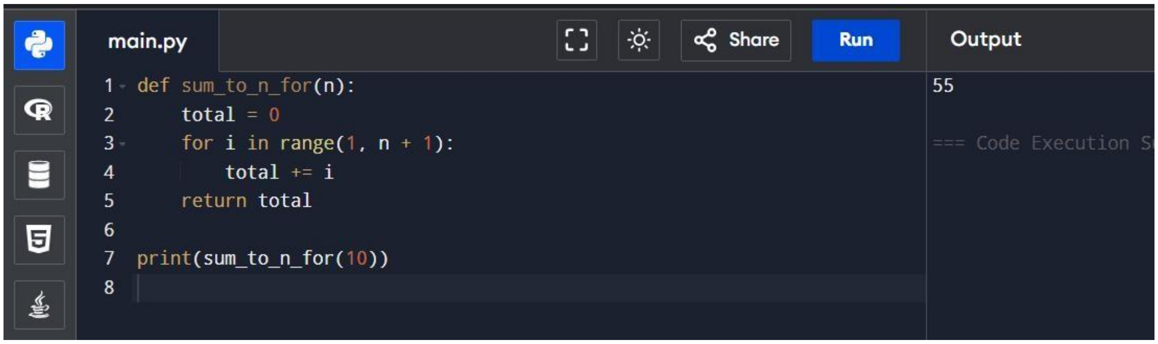
Explanation:
The conditions are checked in sequence. The first matching condition determines the age group.

Task 4: For and While Loops – Sum of First n Numbers

Prompt Used:
"Generate Python code to find sum of first n natural numbers using loops."

Using for loop:

```
def sum_to_n_for(n):
  total = 0
  for i in range(1, n + 1):
    total += i
  return total

print(sum_to_n_for(10))
```



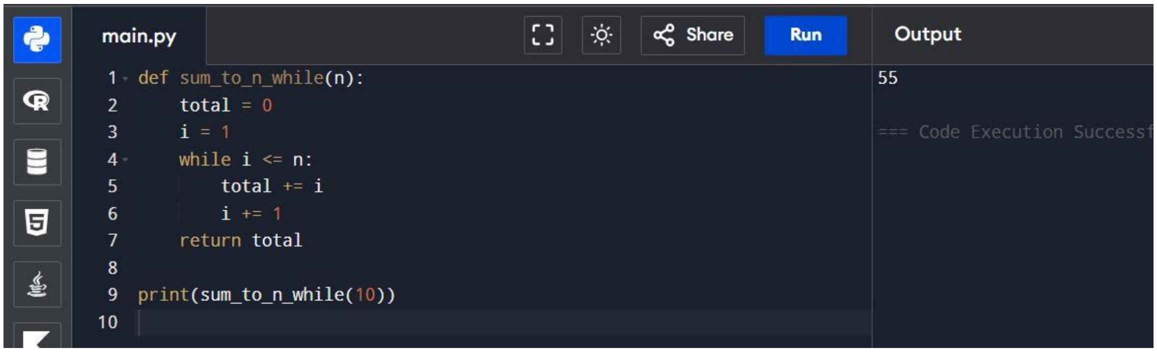
```
def sum_to_n_for(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

print(sum_to_n_for(10))
```

```
55

=== Code Execution S
```

Using while loop:

```
def sum_to_n_while(n):
  total = 0
  i = 1
  while i <= n:
    total += i
    i += 1
  return total

print(sum_to_n_while(10))
```

Output:
55



```
def sum_to_n_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

print(sum_to_n_while(10))
```

```
55

=== Code Execution Successf
```

Analysis:
Both approaches are correct. Loop-based methods are simple, while mathematical formulas can be more efficient.

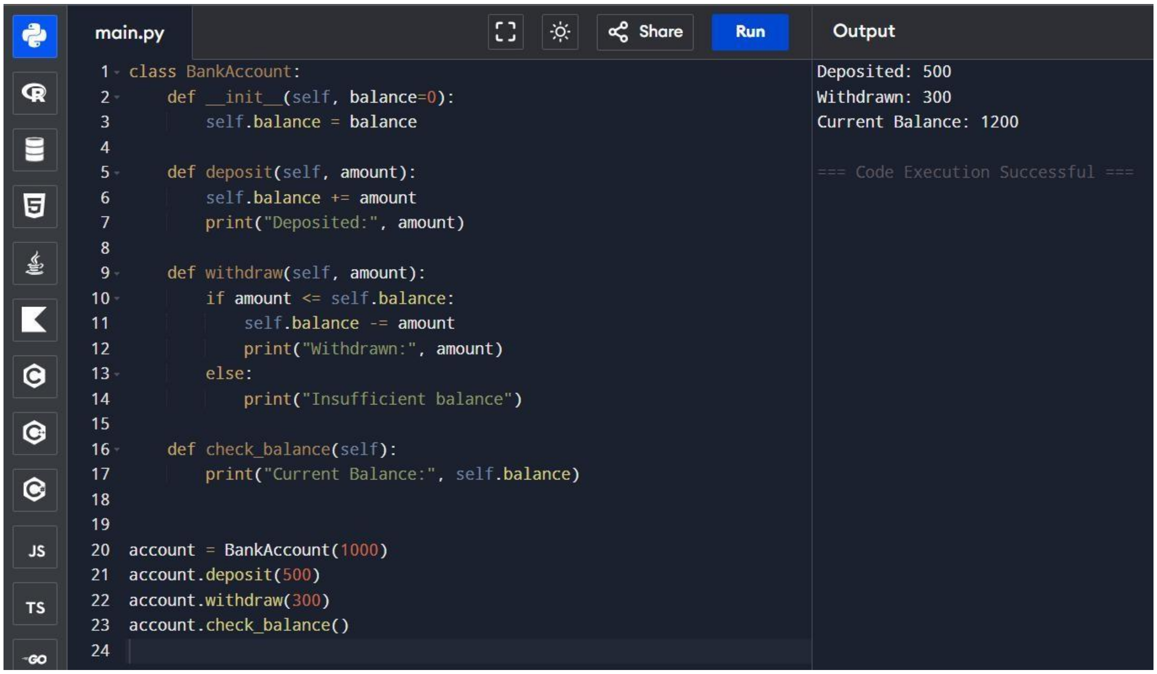
Task 5: Classes – Bank Account Class

Prompt Used:
"Generate a Python BankAccount class with deposit, withdraw, and check balance methods."

```
class BankAccount:
  def _init_(self, balance=0):
    self.balance = balance

  def deposit(self, amount):
    self.balance += amount
    print("Deposited:", amount)

  def withdraw(self, amount):
    if amount <= self.balance:
      self.balance -= amount
      print("Withdrawn:", amount)
    else:
      print("Insufficient balance")

  def check_balance(self):
    print("Current Balance:", self.balance)


account = BankAccount(1000)
account.deposit(500)
account.withdraw(300)
account.check_balance()
```

Output:
Deposited: 500
Withdrawn: 300
Current Balance: 1200

Explanation:
The class maintains account balance and updates it through deposit and withdraw methods.



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

    def deposit(self, amount):
        self.balance += amount
        print("Deposited:", amount)

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print("Withdrawn:", amount)
        else:
            print("Insufficient balance")

    def check_balance(self):
        print("Current Balance:", self.balance)


account = BankAccount(1000)
account.deposit(500)
account.withdraw(300)
account.check_balance()
```

```
Deposited: 500
Withdrawn: 300
Current Balance: 1200

=== Code Execution Successful ===
```

Overall Conclusion

This lab demonstrates how AI-assisted code completion helps in generating structured, readable, and correct Python programs. Human review is essential to ensure correctness and efficiency.