

## ASSIGNMENT-6.3

Name: Y. Poojitha

HT. No: 2303A51499

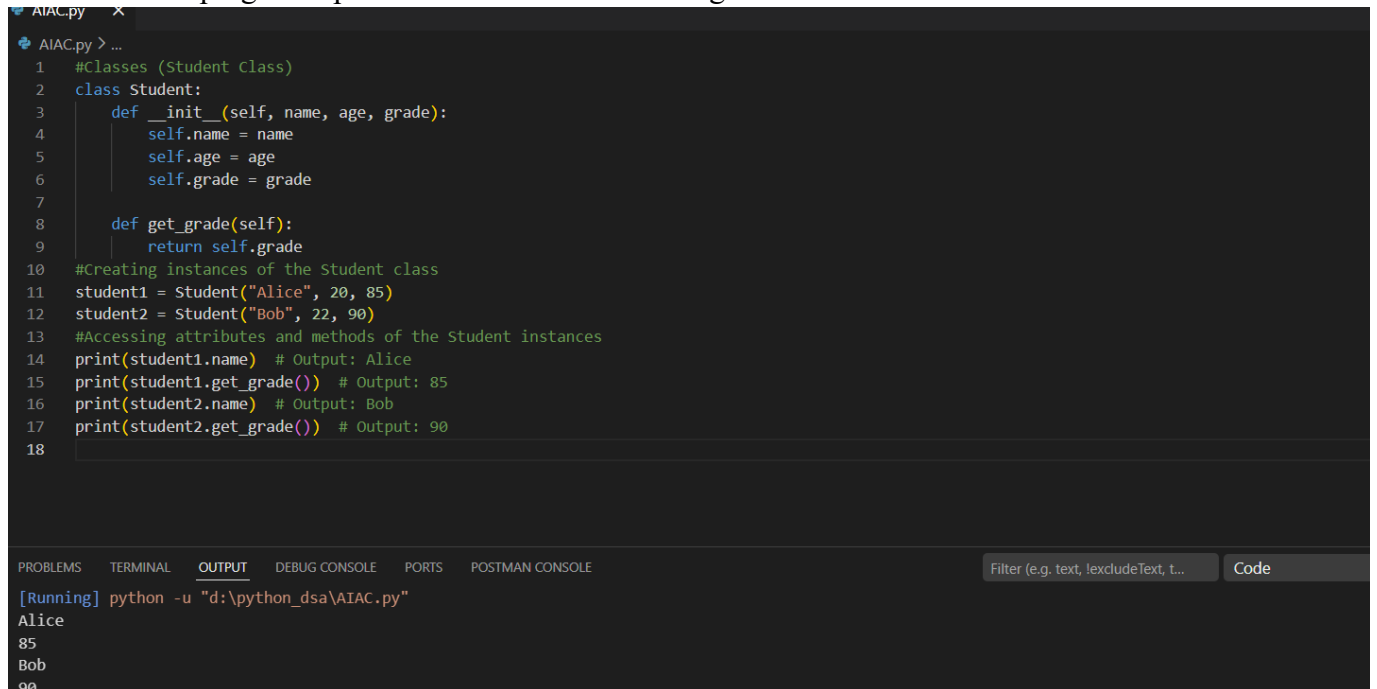
Batch: 08

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

Task Description #1: Classes (Student Class)

Scenario

You are developing a simple student information management module.



```
AIAC.py > ...
1  #Classes (Student Class)
2  class Student:
3      def __init__(self, name, age, grade):
4          self.name = name
5          self.age = age
6          self.grade = grade
7
8      def get_grade(self):
9          return self.grade
10 #Creating instances of the Student class
11 student1 = Student("Alice", 20, 85)
12 student2 = Student("Bob", 22, 90)
13 #Accessing attributes and methods of the Student instances
14 print(student1.name) # Output: Alice
15 print(student1.get_grade()) # Output: 85
16 print(student2.name) # Output: Bob
17 print(student2.get_grade()) # Output: 90
18
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS POSTMAN CONSOLE

[Running] python -u "d:\python\_dsa\AIAC.py"

Alice  
85  
Bob  
90

Task 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).

```

1  #Task2
2  #generate the same functionality using another controlled looping structure
3  def print_multiples(num):
4      for i in range(1, 11):
5          print(num * i)
6  print_multiples(5)
7  def print_multiples_while(num):
8      i = 1
9      while i <= 10:
10         print(num * i)
11         i += 1
12
13  print_multiples_while(5)

```

```

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

```

### Task 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).

```

1 #Task3
2 #to generate nested if-elif-else conditional statements to classify age groups
  (e.g., child, teenager, adult, senior).
3
4 def classify_age(age):
5     if age < 13:
6         return "Child"
7     elif age < 20:
8         return "Teenager"
9     elif age < 60:
10        return "Adult"
11    else:
12        return "Senior"
13
14 print(classify_age(25))

```

Adult

#### Task Description #4: For and While Loops (Sum of First n Numbers)

##### Scenario

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

##### Task

- Use AI assistance to generate a `sum_to_n()` function using a for loop.
- Analyze the generated code.
- Ask the AI to suggest an alternative implementation using a while loop or a mathematical formula.

```

1 #Task4
2 #For and While Loops (Sum of First n Numbers)
3 def sum_to_n(n):
4     total = 0
5     for i in range(1, n + 1):
6         total += i
7     return total
8
9 print(sum_to_n(10))
10 def sum_to_n_while(n):
11     total = 0
12     i = 1
13     while i <= n:
14         total += i
15         i += 1
16     return total
17
18 print(sum_to_n_while(10))

```

55

55

#### 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.

```
1  #Task5
2  #generate a Bank Account class with methods such as deposit(), withdraw(), and
   check_balance().
3
4  class BankAccount:
5      def __init__(self, account_holder, balance=0):
6          self.account_holder = account_holder
7          self.balance = balance
8
9      def deposit(self, amount):
10         self.balance += amount
11         print("Deposited:", amount)
12
13     def withdraw(self, amount):
14         if amount <= self.balance:
15             self.balance -= amount
16             print("Withdrawn:", amount)
17         else:
18             print("Insufficient balance")
19
20     def check_balance(self):
21         print("Current Balance:", self.balance)
22
23
24 # Object creation
25 account = BankAccount("Vyshnavi", 5000)
26 account.deposit(2000)
27 account.withdraw(1500)
28 account.check_balance()

```

Deposited: 2000  
Withdrawn: 1500  
Current Balance: 5500