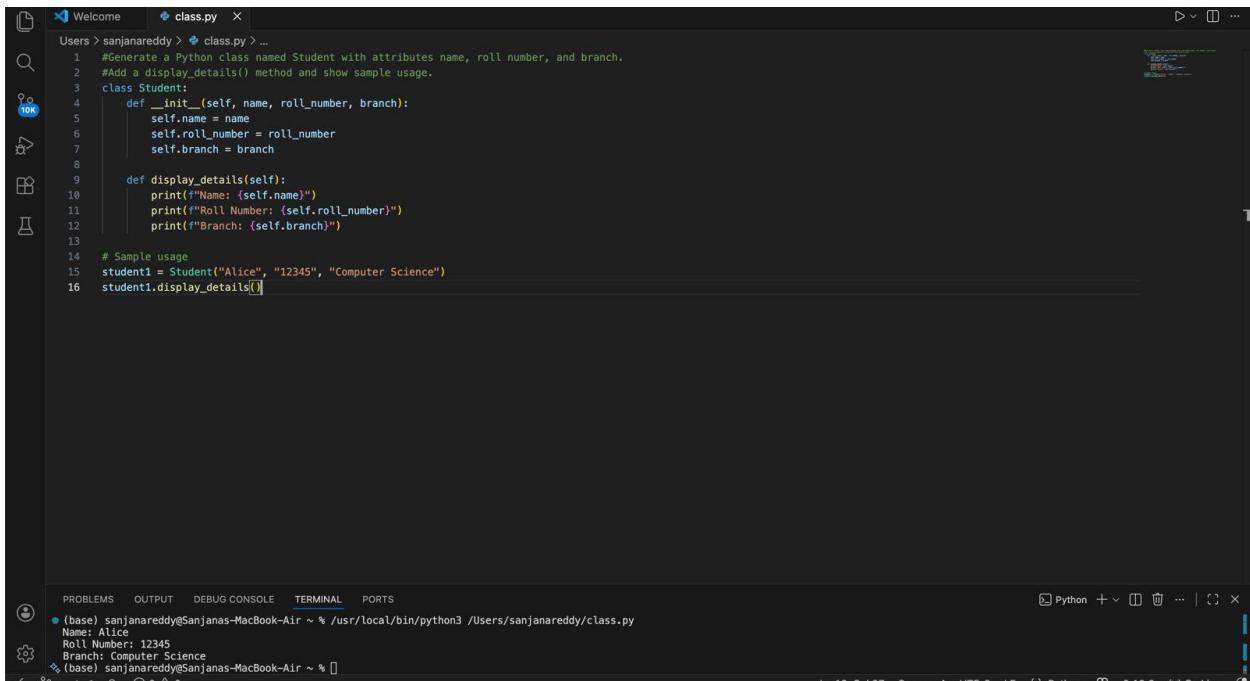


NAME:Boppidi Sanjana Reddy

ROLL NO:2303A51455

BATCH:07

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



The screenshot shows a Visual Studio Code (VS Code) interface. On the left is the code editor with a file named 'class.py'. The code defines a 'Student' class with an __init__ method and a display_details method. It includes a sample usage section. On the right is the terminal window, which shows the output of running the script with Alice as the student information.

```
1 #Generate a Python class named Student with attributes name, roll number, and branch.
2 #Add a display_details() method and show sample usage.
3
4 class Student:
5     def __init__(self, name, roll_number, branch):
6         self.name = name
7         self.roll_number = roll_number
8         self.branch = branch
9
10    def display_details(self):
11        print("Name: " + self.name)
12        print("Roll Number: " + str(self.roll_number))
13        print("Branch: " + self.branch)
14
15 # Sample usage
16 student1 = Student("Alice", 12345, "Computer Science")
17 student1.display_details()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/class.py
Name: Alice
Roll Number: 12345
Branch: Computer Science
% (base) sanjanareddy@Sanjanas-MacBook-Air ~ % []
```

TASK 1 – Classes (Student Class)

Analysis

The AI generated a correct class structure.

The constructor and method usage are clear.

The code is readable and easy to understand.

TASK 2 – Loops (Multiples of a Number)

```
1 #Generate a Python function to print the first 10 multiples of a number using a for loop.
2 def print_multiples(n):
3     for i in range(1, 11):
4         print(f"{n} x {i} = {n * i}")
5
6 # Example usage
7 print_multiples(5)
8 #Generate the same functionality using a while loop.
9 def print_multiples_while(n):
10    i = 1
11    while i <= 10:
12        print(f"{n} x {i} = {n * i}")
13        i += 1
14 # Example usage
15 print_multiples_while(5)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/loops.py
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/loops.py
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % []
```

Analysis

The for loop is simpler and concise.

The while loop provides more control.

Both approaches produce correct output.

TASK 3 – Conditional Statements (Age Classification)

The screenshot shows a code editor interface with a dark theme. In the top navigation bar, there are tabs for 'Welcome', 'conditional.py' (which is the active tab), 'loops.py', and 'class.py'. Below the tabs, the code for 'conditional.py' is displayed:

```
1 #Generate a Python function using if-elif-else to classify age into child, teenager, adult, and senior.
2 def classify_age(age):
3     if age < 0:
4         return "Invalid age"
5     elif age < 13:
6         return "Child"
7     elif age < 20:
8         return "Teenager"
9     elif age < 65:
10        return "Adult"
11    else:
12        return "Senior"
13
14 # Example usage
15 print(classify_age(10)) # Output: Child
16 print(classify_age(16)) # Output: Teenager
17 print(classify_age(30)) # Output: Adult
18 print(classify_age(70)) # Output: Senior
```

Below the code editor, there is a terminal window showing the output of running the script:

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/conditional.py
Child
Teenager
Adult
Senior
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % []
```

Analysis

The conditional logic is clear and correct.

All age groups are handled properly.

TASK 4 – For and While Loops (Sum of First n Numbers)

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'Welcome', 'sumof_natural.py' (which is the active tab), 'conditional.py', 'loops.py', and 'class.py'. Below the tabs, the code for 'sumof_natural.py' is displayed:

```
Users > sanjanareddy > sumof_natural.py > ...
1 #Generate a Python function to calculate the sum of first n natural numbers using a for loop.
2 def sum_natural_numbers(n):
3     total = 0
4     for i in range(1, n + 1):
5         total += i
6     return total
7
8 # Example usage
9 print(sum_natural_numbers(5)) # Output: 15
10 #Generate the same functionality using a while loop.
11 def sum_natural_numbers_while(n):
12     total = 0
13     i = 1
14     while i <= n:
15         total += i
16         i += 1
17     return total
18
19 # Example usage
20 print(sum_natural_numbers_while(7)) # Output: 28
```

Below the code editor, there is a navigation bar with icons for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is currently selected. In the terminal pane, the following command history is shown:

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/sumof_natural.py
15
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/sumof_natural.py
15
28
% (base) sanjanareddy@Sanjanas-MacBook-Air ~ % []
```

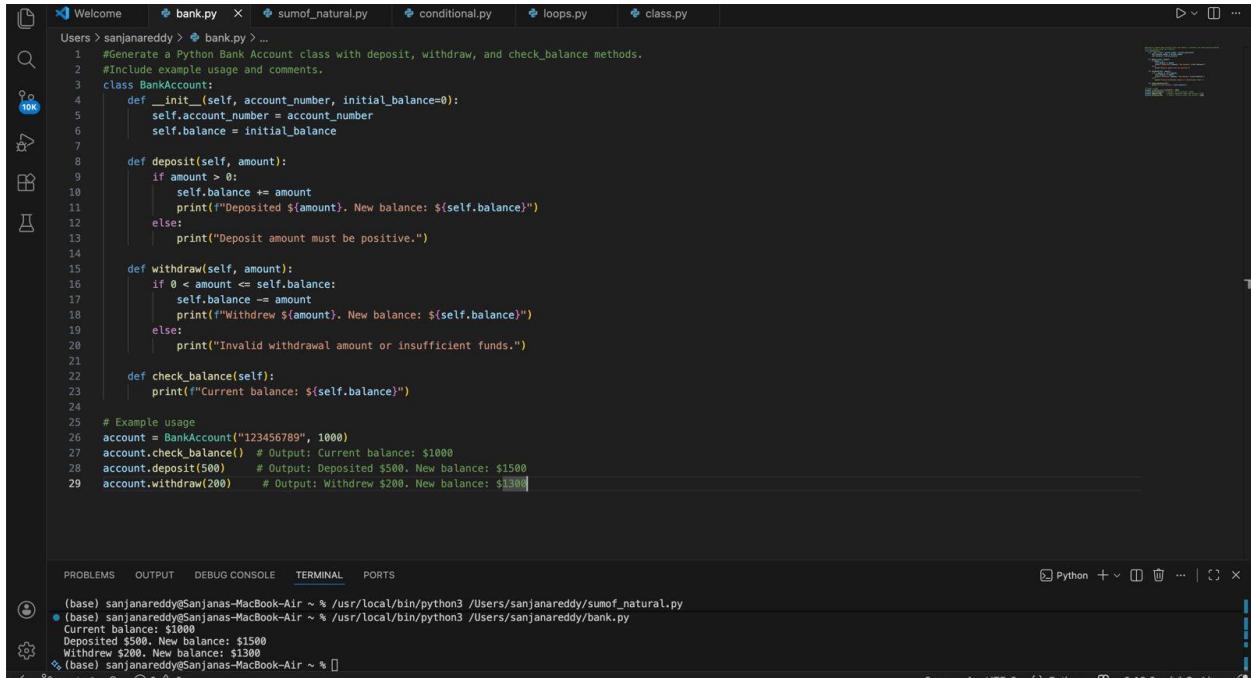
Comparison

The for loop implementation is concise.

The while loop implementation is flexible.

Both approaches give the same result.

TASK 5 – Classes (Bank Account Class)



The screenshot shows a Python code editor in VS Code. The file being edited is `bank.py`. The code defines a `BankAccount` class with methods for depositing and withdrawing funds, and checking the balance. The code is well-formatted with proper indentation and comments. Below the code editor, the terminal window shows the execution of the script and its output.

```
1 #Generate a Python Bank Account class with deposit, withdraw, and check_balance methods.
2 #Include example usage and comments.
3 class BankAccount:
4     def __init__(self, account_number, initial_balance=0):
5         self.account_number = account_number
6         self.balance = initial_balance
7
8     def deposit(self, amount):
9         if amount > 0:
10             self.balance += amount
11             print(f"Deposited ${amount}. New balance: ${self.balance}")
12         else:
13             print("Deposit amount must be positive.")
14
15     def withdraw(self, amount):
16         if 0 < amount <= self.balance:
17             self.balance -= amount
18             print(f"Withdrew ${amount}. New balance: ${self.balance}")
19         else:
20             print("Invalid withdrawal amount or insufficient funds.")
21
22     def check_balance(self):
23         print(f"Current balance: ${self.balance}")
24
25 # Example usage
26 account = BankAccount("123456789", 1000)
27 account.check_balance() # Output: Current balance: $1000
28 account.deposit(500)    # Output: Deposited $500. New balance: $1500
29 account.withdraw(200)   # Output: Withdrew $200. New balance: $1300
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/sumof_natural.py
● (base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/bank.py
Current balance: \$1000
Deposited \$500. New balance: \$1500
Withdrew \$200. New balance: \$1300
% (base) sanjanareddy@Sanjanas-MacBook-Air ~ % []

Explanation

The AI generated a clean object-oriented design.

The methods work correctly and are easy to understand.

The code is readable and efficient.