

Lab Assignment # 05

Course Title : AI Assistant Coding

Name of Student : B.Sai Varsha

Enrollment No. : 2303A54065

Batch No. : 48

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.

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.

Prompt:

Create a Python Student class with attributes name, roll number, and branch.

Add a method `display_details()` to print student information.

Create an object and display the details.

Code :

```
class Student:  
    def __init__(self, name, roll_number, branch):  
        self.name = name  
        self.roll_number = roll_number  
        self.branch = branch  
    def display_details(self):  
        print("Name:", self.name)  
        print("Roll Number:", self.roll_number)  
        print("Branch:", self.branch)  
  
student1 = Student("Rithika", "23CS101", "CSE")  
student1.display_details()
```

Expected Output #1

The screenshot shows a dark-themed instance of Visual Studio Code. In the center-left pane, there is a code editor with a file named 'task1.py' open. The code defines a 'Student' class with an __init__ method to initialize name, roll number, and branch, and a display_details method to print them. An object 'student1' is created and its details are printed. The terminal at the bottom shows the output of running the script, which includes the student's name, roll number, and branch. The status bar at the bottom right indicates the code is in Python mode, line 15, column 42, with 3.9.6 Python installed.

```
1 # Create a Python Student class with attributes name, roll number, and branch.
2 # Add a method display_details() to print student information.
3 # Create an object and display the details.
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(f"Name: {self.name}")
12        print(f"Roll Number: {self.roll_number}")
13        print(f"Branch: {self.branch}")
14 # Creating an object of the Student class
15 student1 = Student("Rithika", "2303a54048", "Computer Science")
16 # Displaying the student details
17 student1.display_details()
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task1.py
Name: Rithika
Roll Number: 2303a54048
Branch: Computer Science
% anumandlarithika@Mac Ai_Assisted_lab %

Ln 15, Col 42 Spaces: 4 LF {} Python 3.9.6

Code Explanation

- The __init__() method initializes student details.
- display_details() prints all attributes.
- An object is created and the method is executed.

Comments / Analysis

- AI generated a correct class structure.
- Code is clear, readable, and follows OOP principles.

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).

Prompt 1:

Write a Python function to print the first 10 multiples of a given number using a for loop.

Code :

```
def print_multiples(num):
    for i in range(1, 11):
        print(num * i)
print_multiples(5)
```

Prompt 2:

Write the same functionality using a while loop.

Code :

```
def print_multiples_while(num):
    i = 1
    while i <= 10:
        print(num * i)
        i += 1
print_multiples_while(5)
```

Expected Output #2

The screenshot shows a dark-themed instance of Visual Studio Code. In the top-left corner, there's a file icon and the text "task2.py". The main editor area contains the following Python code:

```
1  #Write a Python function to print the first 10 multiples of a given number using a for loop.
2  def print_multiples(number):
3      for i in range(1, 11):
4          print(number * i)
5  # Example usage:
6  print_multiples(5)
7
```

Below the editor, the "TERMINAL" tab is selected, showing the output of running the script:

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task2.py
5
10
15
20
25
30
35
40
45
50
```

The bottom status bar indicates the file is saved, the line count is 7, column 1, and the Python version is 3.9.6.

Code Explanation

- for loop iterates fixed number of times.
- while loop uses a condition to control execution.

Comments / Analysis

- Both loops give correct results.
- for loop is simpler and more readable.
- while loop offers flexible control.

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).

Prompt :

Generate nested if-elif-else conditions to classify age as child, teenager, adult, or senior.

Code :

```
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))
```

Expected Output #3

The screenshot shows a code editor interface with a Python file named 'task3.py' open. The code uses nested if-elif-else statements to classify an age input into four categories: child, teenager, adult, or senior. Below the code editor is a terminal window showing the execution of the script and its output for various age inputs.

```
lab6.3 > task3.py > ...
1 #Generate nested if-elif-else conditions to classify age as child, teenager, adult, or senior.
2 age = int(input("Enter your age: "))
3 if age < 13:
4     print("You are a child.")
5 elif 13 <= age < 20:
6     print("You are a teenager.")
7 elif 20 <= age < 65:
8     print("You are an adult.")
9 else:
10    print("You are a senior.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task3.py
Enter your age: 21
You are an adult.
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task3.py
Enter your age: 18
You are a teenager.
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task3.py
Enter your age: 5
You are a child.
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task3.py
Enter your age: 70
You are a senior.
anumandlarithika@Mac Ai_Assisted_lab %
```

Code Explanation

- Conditions are checked in sequence.
- The correct age category is returned.

Comments / Analysis

- AI generated logical and efficient conditions.
- Easy to understand and maintain.

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.

Prompt 1:

Create a function sum_to_n(n) using a for loop.

Code :

```
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
print(sum_to_n(10))
```

Prompt 2 :

Generate the same function using a while loop.

Code :

```
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))
```

Expected Output #4

The screenshot shows a code editor interface with two tabs: 'task4.py' and 'task5.py'. The 'task4.py' tab is active, displaying the following code:

```
1  #Create a function sum_to_n(n) using a for loop.
2  def sum_to_n(n):
3      total = 0
4      for i in range(1, n + 1):
5          total += i
6      return total
7  #Example usage:
8  print(sum_to_n(5)) # Output: 15
9
10 #Generate the same function using a while loop.
11 def sum_to_n_while(n):
12     total = 0
13     i = 1
14     while i <= n:
15         total += i
16         i += 1
17     return total
18 #Example usage:
19 print(sum_to_n_while(5)) # Output: 15
```

The 'task5.py' tab shows a similar structure but with different code. The terminal at the bottom shows the output of both functions:

```
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task4.py
15
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task5.py
15
```

Code Explanation

- Loops accumulate the sum step-by-step.
- Both methods produce the same result.

Comments / Analysis

- for loop is concise.
- while loop is useful for dynamic conditions.

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.

Prompt :

Create a BankAccount class with deposit(), withdraw(), and check_balance() methods.

Add comments and demonstrate usage.

Code :

```
class BankAccount:  
    def __init__(self, holder, balance=0):  
        self.holder = holder  
        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("Rithika", 1000)  
account.deposit(500)  
account.withdraw(300)  
account.check_balance()
```

Expected Output #5

The screenshot shows a code editor interface with a Python file named 'task5.py' open. The code defines a 'BankAccount' class with methods for depositing and withdrawing money, and checking the balance. The code is well-commented and uses f-strings for output. Below the code editor is a terminal window showing the execution of the script and its output. The terminal shows the creation of an account, deposits of \$50 and \$30, and a withdrawal of \$30, with the final balance being \$120.

```
1  #Create a BankAccount class with deposit(), withdraw(), and check_balance() methods.
2  #Add comments and demonstrate usage.
3  class BankAccount:
4      def __init__(self, initial_balance=0):
5          self.balance = initial_balance
6
7      def deposit(self, amount):
8          """Deposit money into the account."""
9          self.balance += amount
10         print(f"Deposited ${amount}. New balance: ${self.balance}")
11
12     def withdraw(self, amount):
13         """Withdraw money from the account."""
14         if amount > self.balance:
15             print("Insufficient funds.")
16         else:
17             self.balance -= amount
18             print(f"Withdrew ${amount}. New balance: ${self.balance}")
19
20     def check_balance(self):
21         """Check the current balance of the account."""
22         print(f"Current balance: ${self.balance}")
23
24 # Demonstrate usage
25 account = BankAccount(100)
26 account.check_balance() # Output: Current balance: $100
27 account.deposit(50)    # Output: Deposited $50. New balance: $150
28 account.withdraw(30)   # Output: Withdrew $30. New balance: $120
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

anumandlarithika@Mac Ai_Assisted_lab % clear
anumandlarithika@Mac Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab6.3/task5.py
Current balance: \$100
Deposited \$50. New balance: \$150
Withdrew \$30. New balance: \$120
anumandlarithika@Mac Ai_Assisted_lab %

Ln 28, Col 67 Spaces: 4 UTF-8 LF () Python 3.9.6

Code Explanation

- Class models real-world bank operations.
- Methods update and display account balance.

Comments / Analysis

- AI generated a realistic and correct class.
- Proper validation is included for withdrawal.

Conclusion

GitHub Copilot effectively assisted in generating correct and readable Python code for classes, loops, and conditionals. The AI tool improved coding speed while maintaining accuracy and clarity.