

# AI Assistant Coding

## Assignment 6.3

Name : Varsha Sri Chukka HT. No : 2303A52494 Batch: 50

### 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.

##### 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.

##### #PROMPT

Write a Python class named `Student` with attributes `name`, `roll_number`, and `branch`. Include a constructor (`__init__`) to initialize these attributes and a method `display_details()` that prints them in a readable format. Also create a sample object and call the method to show the output.

##### #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(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")

# Create a sample object
student1 = Student("Varsha", "2494", "Computer Science")
# Call the method to show the output
student1.display_details()
```

#OUTPUT

```
de files\week6.py
Name: Varsha
Roll Number: 2494
Branch: Computer Science
```

### #Why This Prompt Was chosen

This prompt was chosen because it precisely defines the required class structure, attributes, and method without any ambiguity, ensuring the AI generates exactly what the task asks for. It also explicitly instructs the AI to include a constructor and sample execution, which aligns directly with the assignment's expected output. Additionally, it promotes clear and readable code, making verification and analysis straightforward.

## 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).

## Expected Output #2

- Correct loop-based Python implementation.
- Output showing the first 10 multiples of a number.
- Comparison and analysis of different looping approaches.

### #PROMPT

Write a Python function `print_multiples(n)` that uses a for loop to print the first 10 multiples of a given number `n`. Then rewrite the same functionality using a while loop. Also include sample function calls and expected output

### #CODE

```
def print_multiples_for_loop(n):
    print("Using for loop:")
    for i in range(1, 11):
        print(n * i)
def print_multiples_while_loop(n):
    print("Using while loop:")
    i = 1
    while i <= 10:
        print(n * i)
        i += 1
# Sample function calls
print_multiples_for_loop(5)
print_multiples_while_loop(5)
```

### #OUTPUT

Using for loop:	Using while loop:
5	5
10	10
15	15
20	20
25	25
30	30
35	35
40	40
45	45
50	50

PS C:\Users\VARSHA\OneDrive\

### Why This Prompt Was Chosen

This prompt clearly states the task of printing the first 10 multiples using loops, ensuring correct implementation. It explicitly asks for both for and while loops to enable comparison. It also includes sample execution, making validation simple.

## **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).

### **Expected Output #3**

- A Python function that classifies age into appropriate groups.
- Clear and correct conditional logic.
- Explanation of how the conditions work.

### **#PROMPT**

Write a Python function `classify_age(age)` that uses nested if-elif-else statements to classify a person as child, teenager, adult, or senior. Then implement the same classification using an alternative approach (simplified conditions or dictionary-based logic), and briefly explain how the conditions work.

## #CODE

```
def classify_age_nested(age):
    if age < 13:
        return "Child"
    elif 13 <= age < 20:
        return "Teenager"
    elif 20 <= age < 60:
        return "Adult"
    else:
        return "Senior"
def classify_age_alternative(age):
    age_groups = {
        range(0, 13): "Child",
        range(13, 20): "Teenager",
        range(20, 60): "Adult",
        range(60, 150): "Senior"
    }
    for age_range, classification in age_groups.items():
        if age in age_range:
            return classification
# Sample function calls
print(classify_age_nested(10)) # Output: Child
print(classify_age_nested(15)) # Output: Teenager
print(classify_age_nested(30)) # Output: Adult
print(classify_age_nested(70)) # Output: Senior
print(classify_age_alternative(10)) # Output: Child
print(classify_age_alternative(15)) # Output: Teenager
print(classify_age_alternative(30)) # Output: Adult
print(classify_age_alternative(70)) # Output: Senior
```

## #OUTPUT

```
PS C:\Users\VARSHA
Adult
Senior
Adult
Adult
Senior
PS C:\Users\VARSHA
```

## # Code Explanation

In the alternative approach, we use a dictionary where the keys are ranges representing age groups and the values are their classifications. We iterate through the dictionary and check if the given age falls within any of the defined ranges. This method simplifies the classification logic by avoiding multiple nested conditions and makes it easier to manage age groups.

## #Why This Prompt Was Chosen

This prompt clearly instructs the AI to use conditional logic for age classification, matching the task requirements. It asks for both traditional if-elif-else and an alternative approach, enabling comparison. It also requires an explanation, helping in understanding and analysis.

---

## 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.

### Expected Output #4

- Python function to compute the sum of first n numbers.
- Correct output for sample inputs.
- Explanation and comparison of different approaches.

## #PROMPT

Write a Python function `sum_to_n(n)` that calculates the sum of the first n natural numbers using a for loop. Then provide an alternative implementation using a while loop or the mathematical formula, along with sample outputs.

## #CODE

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

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

def sum_to_n_formula(n):
    return n * (n + 1) // 2

# Sample outputs
print("Sum using for loop:", sum_to_n_for_loop(5))    # Output: 15
print("Sum using while loop:", sum_to_n_while_loop(5)) # Output: 15
print("Sum using formula:", sum_to_n_formula(5))       # Output: 15
```

## #OUTPUT

```
de_files\week6.py*
Sum using for loop: 15
Sum using while loop: 15
Sum using formula: 15
PS C:\Users\VARSHA\OneDrive\Desktop\DevOps> []
```

## #Why This Prompt Was Chosen

This prompt clearly specifies the requirement to compute the sum using a for loop, ensuring correct implementation. It also asks for an alternative method, allowing comparison of approaches. Including sample outputs makes validation and analysis easier.

## 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 explanation.

#### #PROMPT

Create a Python class `BankAccount` with attributes `account_holder` and `balance`. Include methods `deposit(amount)`, `withdraw(amount)`, and `check_balance()`. Demonstrate deposit and withdrawal operations with sample objects, and add meaningful comments explaining the logic.

#### #CODE

```

and check_balance(). Demonstrate deposit and withdrawal operations with sample objects, and add mea
class BankAccount:
    def __init__(self, account_holder, balance=0):
        # Initialize the account holder's name and starting balance
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        # Add the specified amount to the balance
        if amount > 0:
            self.balance += amount
            print(f"Deposited: {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        # Subtract the specified amount from the balance if sufficient funds are available
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: {amount}. New balance: {self.balance}")
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        # Return the current balance
        print(f"Current balance: {self.balance}")

# Demonstrate deposit and withdrawal operations with sample objects
account1 = BankAccount("Varsha", 1000)
account1.check_balance() # Output: Current balance: 1000
account1.deposit(500)    # Output: Deposited: 500. New balance: 1500
account1.withdraw(200)   # Output: Withdrew: 200. New balance: 1300
account1.withdraw(2000)  # Output: Insufficient funds for this withdrawal.
account1.check_balance() # Output: Current balance: 1300
account2 = BankAccount("Chandu")
account2.check_balance() # Output: Current balance: 0
account2.deposit(300)    # Output: Deposited: 300. New balance: 300
account2.withdraw(100)   # Output: Withdrew: 100. New balance: 200
account2.check_balance() # Output: Current balance: 200

```

## #OUTPUT

```

Current balance: 1000
Deposited: 500. New balance: 1500
Withdrew: 200. New balance: 1300
Insufficient funds for this withdrawal.
Current balance: 1300
Current balance: 0
Deposited: 300. New balance: 300
Withdrew: 100. New balance: 200
Current balance: 200
PS C:\Users\VARSHA\OneDrive\Desktop\DevOps> []

```

## **#Why This Prompt Was Chosen**

This prompt clearly defines the required class and its methods, ensuring the AI builds a complete Bank Account model. It explicitly asks for sample operations to verify functionality. It also requires meaningful comments, helping in understanding and analysis of the code.