

ASSIGNMENT 6.3

NAME: Sahasra

2303A52499

BATCH: 50

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:

Generate a python code having python class names students. The class should have attributes: name, roll_number, and branch. Include a constructor to initialize these attributes. Add a method `display_details()` that prints the students information. Create a sample object and display its details.

CODE:

```

## 2303A52499

# Generate a python code having python class names students . The class should have attributes: name,roll_number, and branch.
# Include a constructor to initialize these attributes. Add a method display_details() that prints the students information.
# Create a sample object and display its details.

class Students:
    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}")

```

OUTPUT:

```

.2025121701-win32-x64\bundled\libs\debugpy\launcher' '58323' '--' 'C:\Users\prash\AIAC 6.3.py'
Name: Prashanthi
Roll Number: 2303A52495
Branch: Computer Science
PS C:\Users\prash> 

```

EXPLANATION:

A class is a blueprint used to create objects that group related data and functions together. It defines attributes (data) and methods (behavior) of an entity. Objects are instances of a class that hold actual values. Using classes improves code organization, reusability, and clarity.

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 that prints the first 10 multiples of a given number using a for loop.
Generate the same program using a while loop instead of a for loop.

CODE:

```
# Using a for loop
def print_multiples_for_loop(number):
    print(f"First 10 multiples of {number} using for loop:")
    for i in range(1, 11):
        print(number * i)
print_multiples_for_loop(3)
# Using a while loop
def print_multiples_while_loop(number):
    print(f"First 10 multiples of {number} using while loop:")
    i = 1
    while i <= 10:
        print(number * i)
        i += 1
print_multiples_while_loop(9)
```

OUTPUT:

```
First 10 multiples of 3 using for loop:
3
6
9
12
15
18
21
24
27
30
First 10 multiples of 9 using while loop:
9
18
27
36
45
54
63
72
81
90
PS C:\Users\prash> □
```

EXPLANATION:

This task focuses on using looping constructs to repeatedly execute a set of instructions. It demonstrates how loops can generate the first 10 multiples of a given number efficiently. By implementing both for and while loops, the task highlights different ways to control repetition in Python. It also helps compare readability and control between looping structures.

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 code to generate nested if-elif-else conditional statement to classify age groups eg: child,teenager,adult,senior categories. and also analyse this logic and conditions. and also generate the same age classification using simplified conditions or dictionary based logic.

CODE

```

def classify_age(age):
    if age < 13:
        return "Child"
    elif 13 <= age < 20:
        return "Teenager"
    elif 20 <= age < 60:
        return "Adult"
    else:
        return "Senior"
# Using the function
age = 25
category = classify_age(age)
print(f"Age: {age}, Category: {category}")
# Simplified using dictionary based logic
# Define age categories
age_categories = { "Child": range(0, 13),
                    "Teenager": range(13, 20),
                    "Adult": range(20, 60),
                    "Senior": range(60, 150) }
def classify_age_dict(age):
    for category, age_range in age_categories.items():
        if age in age_range:
            return category
# Using the function
age = 70
category = classify_age_dict(age)
print(f"Age: {age}, Category: {category}")

```

OUTPUT

```

D:\21701_WIN10_2_X64\Bandicam (1105 DebugPy (launched).mp4 - 37.21s
Age: 25, Category: Adult
Age: 70, Category: Senior
PS C:\Users\prash> []

```

EXPLANATION:

This concept uses conditional statements to make decisions based on age values. The program checks age ranges using if-elif-else and assigns the appropriate category. Conditions are evaluated sequentially until one is satisfied. This helps implement clear and logical classification rules in a program.

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. Generate an alternative solution to find the sum of first n natural numbers using a while loop or a mathematical formula. **CODE**

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

# Using a while loop
def sum_to_n_while_loop(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

# Using mathematical formula
def sum_to_n_formula(n):
    return n * (n + 1) // 2

# Testing the functions
n = 10
print(f"Sum of first {n} natural numbers using for loop: {sum_to_n_for_loop(n)}")
print(f"Sum of first {n} natural numbers using while loop: {sum_to_n_while_loop(n)}")
print(f"Sum of first {n} natural numbers using formula: {sum_to_n_formula(n)}")
```

OUTPUT

```
Sum of first 10 natural numbers using for loop: 55
Sum of first 10 natural numbers using while loop: 55
Sum of first 10 natural numbers using formula: 55
PS C:\Users\prash> []
```

EXPLANATION

This concept focuses on calculating the sum of natural numbers using repetition and logic. Looping constructs (for and while) add each number step by step until n . An alternative mathematical formula computes the result directly without iteration. Comparing these approaches highlights efficiency and control in programming.

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.

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

Prompt:

Create a Python class named BankAccount with methods deposit(), withdraw(), and check_balance(). The class should store account holder name and balance. Demonstrate deposit and withdrawal operations.

CODE

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

    def deposit(self, amount):
        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):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: {amount}. New balance: {self.balance}")
        else:
            print("Insufficient balance or invalid withdrawal amount.")

    def check_balance(self):
        print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")

# Demonstrating the BankAccount class
account = BankAccount("John", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(200)
account.check_balance()
```

OUTPUT

```
Account Holder: John, Balance: 1000
Account Holder: John, Balance: 1000
Deposited: 500. New balance: 1500
Withdrew: 200. New balance: 1300
Account Holder: John, Balance: 1300
PS C:\Users\prash> []
```

EXPLANATION

This concept uses a class to model a real-world bank account with data and behavior grouped together. Methods like deposit and withdraw update the account balance based on user actions. Conditional checks ensure valid transactions and prevent errors. This demonstrates practical use of object-oriented programming for real-life applications.