

2303A51504

BATCH-25

Assignment – 6.3

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.

```

1 # Student class
2 class Student:
3     def __init__(self, name, roll_number, branch):
4         self.name = name
5         self.roll_number = roll_number
6         self.branch = branch
7
8     def display_details(self):
9         print("Student Name:", self.name)
10        print("Roll Number:", self.roll_number)
11        print("Branch:", self.branch)
12
13
14 # Creating a Student object
15 student1 = Student("Mouna", 101, "Computer Science")
16
17 # Displaying student details
18 student1.display_details()
19

```

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.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "AI ASTDED" folder, including "ass 6.3.py", "add.py", "app.py", "ass 3.4 ai.pdf", "ass 4.4.docx", "ass 4.4.pdf", "ass 4.4.py", "lab 4.3 word.docx", "lab 4.3 word.pdf", "lab 4.3.py", "lab ass 3.4.py", "lab assignment 3.3.pdf", "lab assignment-1.4.pdf", "lab assignment-2.3.pdf", "lab assignment54 ai.pdf", "leap_year.py", "logger.py", "multiples.py", "README.md", "requirements.txt", and "student.py".
- Code Editor:** The file "ass 6.3.py" is open, containing the following Python code:

```
# Using for loop
def print_multiples_for(n):
    for i in range(1, n+1):
        print(n * i)

print("Using for loop:")
print_multiples_for(5)

# Using while loop
def print_multiples_while(n):
    i = 1
    while i <= n:
        print(n * i)
        i += 1

print("\nUsing while loop:")
print_multiples_while(5)
```
- Terminal:** Shows the command "python" followed by the path "c:\Users\parva\OneDrive\Desktop\AI Astded\multiples.". It also shows a list of files: "powershell", "Python", "Python", "Python", "Python", "powershell", "Python", "Python", "Python". Below this, it says "2 files changed +78 -2 Keep Undo".
- Chat:** A sidebar titled "SIMPLE PYTHON STUDENT CLASS PRO..." with the following text:

use this to multiples of a given number using a for loop.
The function should take the number as a parameter and print each multiple on a new line.
After that, generate the same functionality using a while loop instead of a for loop.
Keep the code simple, correct, and beginner-friendly, and show sample output for a number like 5.
- Bottom Status Bar:** Shows "Ln 37, Col 10", "Spaces: 4", "UTF-8", "CRLF", "Python", "3.13.7", and the current time "9:30 AM".

This screenshot is nearly identical to the first one, showing the same code editor content and terminal output. The terminal now shows the execution of the script "ass 6.3.py" and its output:

```
python
c:\Users\parva\OneDrive\Desktop\AI Astded\multiples.
> Add multiples.py with for/while funct...
> 2 files changed +78 -2 Keep Undo
> + ass 6.3.py
View All Edits
```

The terminal also displays the message "Describe what to build next" and "Agent Auto".

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.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar with a tree view of files and folders, including 'AI Assisted' and 'ass 6.3.py'. The main area is a code editor with the following content:

```
ass 6.3.py
36
37 print("\nUsing while loop:")
38 print_multiples_while(5)
39
40
41 def classify_age(age):
42     if age < 13:
43         print("Child")
44     elif age >= 13 and age <= 19:
45         print("Teenager")
46     elif age >= 20 and age <= 59:
47         print("Adult")
48     else:
49         print("Senior")
50
51
52 # Example usage
53 classify_age(25) # Output: Adult
```

To the right of the code editor is a 'CHAT' panel titled 'SIMPLE PYTHON STUDENT CLASS PRO...'. It contains a message from an AI asking to generate a Python function for classifying age based on nested if-else statements. The user's response is shown below the message. The AI also provides a note about updating the TODO list and creating a classifier.

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.

The screenshot shows the Visual Studio Code interface with the AI Assistant extension open. The Explorer sidebar on the left lists files including 'ass 6.3.py' (marked as modified), 'sum_to_n.py', and other assignment-related files. The main code editor shows the following Python code:

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

# Example usage
print(sum_to_n(10))
```

To the right of the code editor is a large AI-generated task card:

Generate a Python function named `sum_to_n()` that calculates the sum of the first n natural numbers using a for loop. The function should take n as input and return the sum. Keep the code simple, readable, and beginner-friendly.

Managing tasks and outputs: I'll update the todo list and create `sum_to_n.py`. Added 3 todos. Now creating `sum_to_n.py` with the requested function and a demo.

Edited: `sum_to_n.py` +32 -1 Starting: Run `sum_to_n.py` and verify output (6/7)

The terminal at the bottom shows command-line history related to the AI interaction:

```
PS C:\Users\parva\OneDrive\Desktop\AI Asst& C:/Users/parva/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/parva/OneDrive/Desktop/AI Asst/ass 6.3.py"
Adult
PS C:\Users\parva\OneDrive\Desktop\AI Asst& C:/Users/parva/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/parva/OneDrive/Desktop/AI Asst/ass 6.3.py"
Adult
PS C:\Users\parva\OneDrive\Desktop\AI Asst& C:/Users/parva/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/parva/OneDrive/Desktop/AI Asst/ass 6.3.py"
55
PS C:\Users\parva\OneDrive\Desktop\AI Asst>
```

ask 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.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the workspace, including `ass 6.3.py`, `AI Asst`, `addipy`, `age_classifier.py`, `AI ass Ass 1.pdf`, `app.py`, `ass 3.4 ai.pdf`, `ass 4.4.pdf`, `ass 4.4py`, `ass 6.3py`, `lab 4.3 word.docx`, `lab 4.3 word.pdf`, `lab 4.3 py`, `lab 4.3 34.py`, `lab assignment 3.3.pdf`, `lab assignment-1.4.pdf`, `lab assignment-2.3.pdf`, `lab assignment54 ai.pdf`, `leap_year.py`, `logger.py`, `multiples.py`, `README.md`, `requirements.txt`, `student.py`, and `sum_to_n.py`.
- Code Editor:** Displays the `ass 6.3.py` file content. The code defines a `BankAccount` class with methods `deposit`, `withdraw`, and `check_balance`. It also includes an example usage block.
- Terminal:** Shows the command-line output of running the script, demonstrating the deposit and withdrawal operations.
- Chat Window:** Shows a conversation with AI Asst. The user asks to generate a Python class for a basic banking application. The AI responds with requirements and provides a template for the `bank_account.py` file.
- Status Bar:** Shows the current file is `ass 6.3.py`, the line count is 101, column 1, and the date is 2/4/2026.

```

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

    # Method to deposit money
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print("Deposited:", amount)
        else:
            print("Invalid deposit amount")

    # Method to withdraw money
    def withdraw(self, amount):
        if amount < self.balance:
            self.balance -= amount
            print("Withdrawn:", amount)
        else:
            print("Insufficient balance")

    # Method to check current balance
    def check_balance(self):
        print("Current Balance:", self.balance)

# Example usage
account = BankAccount("Mouna", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(300)
account.check_balance()

```