

Siddhartha Reddy Kotha
2303A51491
batch 25

Assignment - 6.3

Date:4/2/26

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 Student class with name, `roll_number`, branch, a constructor, `display_details()` method, and sample object creation with output.

Code

The screenshot shows a code editor interface with a sidebar containing various files and a main workspace for editing a Python script named `lab assignment 6.3.py`. The script defines a `Student` class with `__init__` and `display_details` methods. It then creates a `student1` object and prints its details. The terminal window below shows the execution of the script and its output, which includes the student's name, roll number, and branch.

```

#Generate a Python Student class with name, roll_number, branch, a constructor, di
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}")
student1 = Student("John", 123456, "Computer Science")
student1.display_details()
print(student1.name)
print(student1.roll_number)
print(student1.branch)

```

Output:

The screenshot shows the terminal window from the code editor displaying the output of the `lab assignment 6.3.py` script. The output includes the student's name, roll number, and branch, printed in a readable format.

```

Name: John
Roll Number: 123456
Branch: Computer Science
John
123456
Computer Science

```

Overall explanation:

This program defines a `Student` class to represent student details like name, roll number, and branch.

The constructor (`__init__`) initializes these values when a new object is created.

The `display_details()` method prints all the student information in a readable format.

Finally, a `Student` object is created and its data is accessed using both the method and direct attributes.

Task 2:

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.

Prompt1:

Write a Python function to print the first 10 multiples of a given number using a for loop with sample input and output.

Prompt2:

Generate the same Python function to print the first 10 multiples of a number using a while loop and explain the logic

code:

```
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} * {i} = {number * i}")
print_multiples(5)
#Write a Python function to print the first 10 multiples of a given number
def print_multiples(number):
    i = 1
    while i <= 10:
        print(f"{number} * {i} = {number * i}")
        i += 1
print_multiples(5)
#Write a Python function to print the first 10 multiples of a given number
```

Output:

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a folder named "AI_ASSISTANT_CODING" containing various files: Assignment-4.4.txt, day1.py, file_operations.py, lab assignment 3.3.pdf, lab assignment 4.3.pdf, lab assignment 5.4.pdf, lab assignment 6.3.py (selected), lab assignment-1.4.docx, lab assignment-1.4.pdf, lab assignment-1.docx, lab assignment-1.pdf, lab assignment-2.3.docx, lab assignment-2.3.pdf, lab assignment-3.4.docx, lab assignment-3.4.pdf, lab assignment-5.4.docx, Lab.Assignment_4.3.docx, Lab.Assignment_4.4.docx, Lab-6.3.docx, machine-readable-business-employment-da..., and sample.txt.
- Code Editor:** Displays the content of "lab assignment 6.3.py". The code is as follows:

```

15 print(student1.branch)"""
    
```
- Terminal:** Shows the command "python.exe" running in a terminal window, outputting the multiplication table of 5 from 1 to 10.
- Right Panel:** Includes sections for "Agents" (with a search bar and "New Agent" button) and "Ports" (listing powershell, PowerShell, and Python).

Description:

Both functions correctly print the first 10 multiples of the given number in a neat format.

The loop control (range(1,11) in for and i <= 10 in while) is used properly.

Using print_multiples(5) as sample input clearly demonstrates the expected output.

Task3:

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

Prompt1:

Create a Python function `classify_age(age)` using nested if-elif-else to classify child, teenager, adult, and senior with examples.

Prompt2:

Rewrite the age classification program using a simplified or dictionary-based conditional approach and explain it.

Code:

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

- File Explorer:** Shows various files including AI_ASSISTANT_CODING, ass-1.py, assignment-4.4.docx, assignment-4.4.pdf, assignment-2.4.py, Assignment-4.4.py, Assignment-4.4.txt, day1.py, file_operations.py, lab assignment 3.3.pdf, lab assignment 4.3.pdf, lab assignment 5.4.pdf, lab assignment 6.3.py, lab assignment 1.4.docx, lab assignment 1.4.pdf, lab assignment 1.docx, lab assignment-1.pdf, lab assignment-2.3.docx, lab assignment-2.3.pdf, lab assignment-3.4.docx, lab assignment-3.4.pdf, lab assignment-5.4.docx, Lab Assignment_4.3.docx, Lab Assignment_4.4.docx, lab-6.3.docx, machine-readable-business-employment-da..., sample.txt.
- Code Editor:** Displays a Python script named 6.3.py with the following code:

```

# Create a Python function classify_age(age) using nested if-elif-else to classify child, teenager, adult, and senior
def classify_age(age):
    if age < 13:
        return "child"
    elif age < 18:
        return "teenager"
    elif age < 60:
        return "adult"
    else:
        return "senior"

print(classify_age(10))
print(classify_age(15))
print(classify_age(20))
print(classify_age(65))

```

A comment at the bottom states: "#how-to: the age classification program using a simplified or dictionary-based conditional approach and explain".
- Terminal:** Shows the command: PS D:\VAL_ASSISTANT_CODING & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
- Output:** Displays the output of the script:

```

child
teenager
adult
senior

```
- Bottom Status Bar:** Shows the date (2/4/2026), time (9:43 AM), battery level (ENG IN), and signal strength.

Output:

The screenshot shows a terminal window with the following details:

- File Explorer:** Same as the VS Code interface.
- Terminal:** Shows the command: PS D:\VAL_ASSISTANT_CODING & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
- Output:** Displays the output of the script:

```

child
teenager
adult
senior

```
- Bottom Status Bar:** Shows the date (2/4/2026), time (9:43 AM), battery level (ENG IN), and signal strength.

Description:

Instead of multiple if–elif conditions, age groups are stored in a dictionary where each key is a label and each value is a range of ages.

The function loops through the dictionary and checks which range the given age belongs to.

As soon as a match is found, the corresponding classification is returned.

This approach is more readable, easier to update, and avoids long conditional chains.

Task 4:

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.

Prompt1:

Write a Python function `sum_to_n(n)` to calculate the sum of first n natural numbers using a for loop with sample output.

Prompt2:

Generate an alternative implementation of `sum_to_n(n)` using a while loop or mathematical formula and compare approaches.

Code:

```
#Write a Python function sum_to_n(n) to calculate the sum of first n natural numbers using a for loop with sample output.
def sum_to_n(n):
    sum = 0
    for i in range(1, n+1):
        sum += i
    return sum
print(sum_to_n(10))

#Write a Python function sum_to_n(n) to calculate the sum of first n natural numbers using a while loop with sample output.
def sum_to_n(n):
    sum = 0
    i = 1
    while i <= n:
        sum += i
        i += 1
    return sum
print(sum_to_n(10))
```

Output:

```
PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
child
teenager
adult
senior
child
teenager
adult
senior
1
55
PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
55
PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai te ja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
1
55
PS D:\AI_ASSISTANT_CODING>
```

Description:

Both **age classification approaches** (if–elif and dictionary-based) are implemented properly and give the same correct results (child, teenager, adult, senior). The `sum_to_n(n)` function using a **for loop** correctly adds numbers from 1 to n. The second `sum_to_n(n)` using a **while loop** is also logically correct and produces the same output. One small improvement: since both functions have the **same name**, the second definition overrides the first—this is fine for learning, but in practice you'd use different names like `sum_to_n_for` and `sum_to_n_while`.

Task 5:

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:

Generate a Python `BankAccount` class with `deposit()`, `withdraw()`, `check_balance()` methods, sample usage, and updated balance output.

Code:

A screenshot of a Windows desktop environment. The taskbar at the bottom shows icons for File Explorer, Task View, Start, and several pinned applications including Microsoft Edge, File Explorer, and a Python-related application. The main window is a code editor with tabs for '2.3.py' and 'lab assignment 6.3.py x'. The code in 'lab assignment 6.3.py' is as follows:

```
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
        return f"Deposited {amount}. New balance: {self.balance}"
    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient balance"
        self.balance -= amount
        return f"Withdrew {amount}. New balance: {self.balance}"
    def check_balance(self):
        return f"Current balance: {self.balance}"
account = BankAccount(1000)
print(account.deposit(500))
print(account.withdraw(200))
print(account.check_balance())

```

The terminal window below the code editor shows command-line output:

```
PS D:\AI_ASSISTANT_CODING & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab_assignment 6.3.py"
1
PS D:\AI_ASSISTANT_CODING & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab_assignment 6.3.py"
Deposited 500. New balance: 1500
Withdrew 200. New balance: 1300
Current balance: 1300
PS D:\AI_ASSISTANT_CODING
```

Output:

lab assignment-3.4.pdf
lab assignment-5.4.docx
Lab_Assignment_4.3.docx
Lab_Assignment_4.4.docx
Lab-6.3.docx
machine-readable-business-employment-da...
sample.txt

> OUTLINE
> TIMELINE
✗ ALASSISTANT_CODING ② 0 0

Problems Output Debug Console Terminal Ports

55

- PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"
- 1
- PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 6.3.py"

Deposited 500. New balance: 1500
Withdrew 200. New balance: 1300
Current balance: 1300

- PS D:\AI_ASSISTANT_CODING>

Ctrl+K to generate command

Cursor Tab Search Ln 75, Col 1

Description:

The `BankAccount` class uses a **constructor** to initialize the account with an initial balance. The `deposit()` method correctly adds money to the balance and returns a confirmation message. The `withdraw()` method safely checks for **insufficient balance** before deducting the amount. The `check_balance()` method neatly displays the current balance, and the sample object usage proves all methods work as expected.