

Assignment-6.3

H.NO:2303A54038

Batch:48

1.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.

Code & Output:

The screenshot shows a code editor window with a dark theme. The file tab shows "Lab6.py". The code itself is as follows:

```
Lab6.py
6
7
8 class Student:
9     def __init__(self, name, roll_number, branch):
10         # Initializing attributes
11         self.name = name
12         self.roll_number = roll_number
13         self.branch = branch
14
15     def display_details(self):
16         # Method to display student information
17         print("Student Details:")
18         print(f"Name : {self.name}")
19         print(f"Roll Number: {self.roll_number}")
20         print(f"Branch : {self.branch}")
21
22
23 # Example usage:
24 student1 = Student("Hasitha", "101", "Computer Science")
25 student2 = Student("Ravi", "102", "Mechanical Engineering")
26
27 # Display details of students
28 student1.display_details()
29 print() # Just for spacing
30 student2.display_details()
31
```

Below the code editor is a terminal window showing the execution of the script and its output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
Name : Ravi
Roll Number: 102
Branch : Mechanical Engineering
PS C:\Users\Asus\OneDrive\Desktop\AIAC>
```

A sidebar on the right shows a tree view of Python files in the project.

Prompt: You are in charge of developing a simple student information management

module.create a

student class while making sure the class includes attributes such as name, roll number and branch. Include a method display_details() to print student information. Create all this using python

Code-Explanation: the student class consists of attributes such as Name, Roll number, Branch. Multiple objects such as student1 and student2 are created and their details are accessed outside the class using display_details() method.

2.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).

code & Output:

```
Click to add a breakpoint
33 #2.
34
35 # Case 1: Using a for loop
36 def print_multiples_for(num):
37     print(f"\nFirst 10 multiples of {num} using for loop:")
38     for i in range(1, 11): # Loop from 1 to 10
39         print(f"{num} x {i} = {num * i}")
40
41
42 # Case 2: Using a while loop
43 def print_multiples_while(num):
44     print(f"\nFirst 10 multiples of {num} using while loop:")
45     i = 1
46     while i <= 10: # Loop until i reaches 10
47         print(f"{num} x {i} = {num * i}")
48         i += 1
49
50
51 # Example usage:
52 print_multiples_for(5)
53 print_multiples_while(5)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... X
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
First 10 multiples of 5 using for loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
First 10 multiples of 5 using while loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
First 10 multiples of 5 using for loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
First 10 multiples of 5 using while loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
```

Prompt: you are writing a utility function to display multiples of a given number.make sure to print the first 10 multiples of a given number using a loop.print 2 cases where in one case you use a for loop and in one case you use a while loop to execute it **Code Explanation:** Here,For loop version: Iterates directly over numbers 1–10 using range(1, 11).Where as While loop version Starts with i = 1 and increments until i <= 10..Both prints the same values.

3.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).

code & output:

```

42  #!/usr/bin/python3
43  #3.
44  def classify_age_nested(age):
45      """Return age group using nested if/elif/else."""
46      if age < 0:
47          return "invalid age"
48      else:
49          if age <= 12:
50              return "child"
51          elif age <= 19:
52              return "teenager"
53          else:
54              if age <= 64:
55                  return "adult"
56              else:
57                  return "senior"
58
59  # Examples
60  for a in [5, 16, 30, 70, -1]:
61      print(a, ":", classify_age_nested(a))
62
63  #dictionary-based logic
64  def classify_by_ranges(age):
65      ranges = {
66          (0, 12): "child",
67          (13, 19): "teenager",
68          (20, 64): "adult",
69          (65, float("inf")): "senior",
70      }
71      if age < 0:
72          return "invalid age"
73      for (lo, hi), label in ranges.items():
74          if lo <= age <= hi:
75              return label
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90

```

← AGE GROUP CLASSIFICATION USING CONDITIONAL STATEMENTS
return age group using nested if-elif-else

```

if age < 0:
    return "invalid age"
else:
    if age <= 12:
        return "child"
    elif age <= 19:
        return "teenager"
    else:
        if age <= 64:
            return "adult"
        else:
            return "senior"

```

examples

```

'a in [5, 16, 30, 70, -1]:  
print(a, ":", classify_age_nested(a))

```

2) Dictionary-based alternatives 🎯

a) Range keys lookup

```

def classify_by_ranges(age):
    ranges = {
        (0, 12): "child",
        (13, 19): "teenager",
        (20, 64): "adult",
        (65, float("inf")): "senior"
    }

```

Describe what to build next

Agent Auto

```

● PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
5 → child
16 → teenager
30 → adult
70 → senior
-1 → invalid age
○ PS C:\Users\Asus\OneDrive\Desktop\AIAC>

```

Prompt:

imagine you are building a classification system based on age.use nested if-elif-else control statements to classify age groups(child,teenager,adult,senior).Also provide the same classification using alternative conditional structures such as dictionary-based logic.

code-Explanation

Nested if-else are simple and fast.you can use them when a problem demands fixed rules whereas dictionary based approach is more flexible and easier to chnage but it is slightly slower because it iterates and evaluates functions

4.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.

code & output:

```

Lab6.py
92
93 def print_sum_natural(n):
94     """Print the sum of the first n natural numbers (1..n)."""
95     try:
96         n = int(n)
97     except (TypeError, ValueError):
98         print("Invalid input: n must be an integer")
99         return
100    if n < 0:
101        print("Invalid input: n must be non-negative")
102        return
103
104    total = 0
105    for i in range(1, n + 1):
106        total += i
107    print(total)
108
109
110 if __name__ == "__main__":
111     try:
112         n = input("Enter n: ")
113     except EOFError:
114         print("No input received. Exiting.")
115     else:
116         print_sum_natural(n)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
File "c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py", line 109, in <module>
    n=input()
ValueError: invalid literal for int() with base 10: '& C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py'
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/OneDrive/Desktop/AIAC/Lab6.py
Enter n: 10
55
○ PS C:\Users\Asus\OneDrive\Desktop\AIAC>

```

Prompt:

you are building a function to print the sum of n natural numbers.use a for loop to get the output

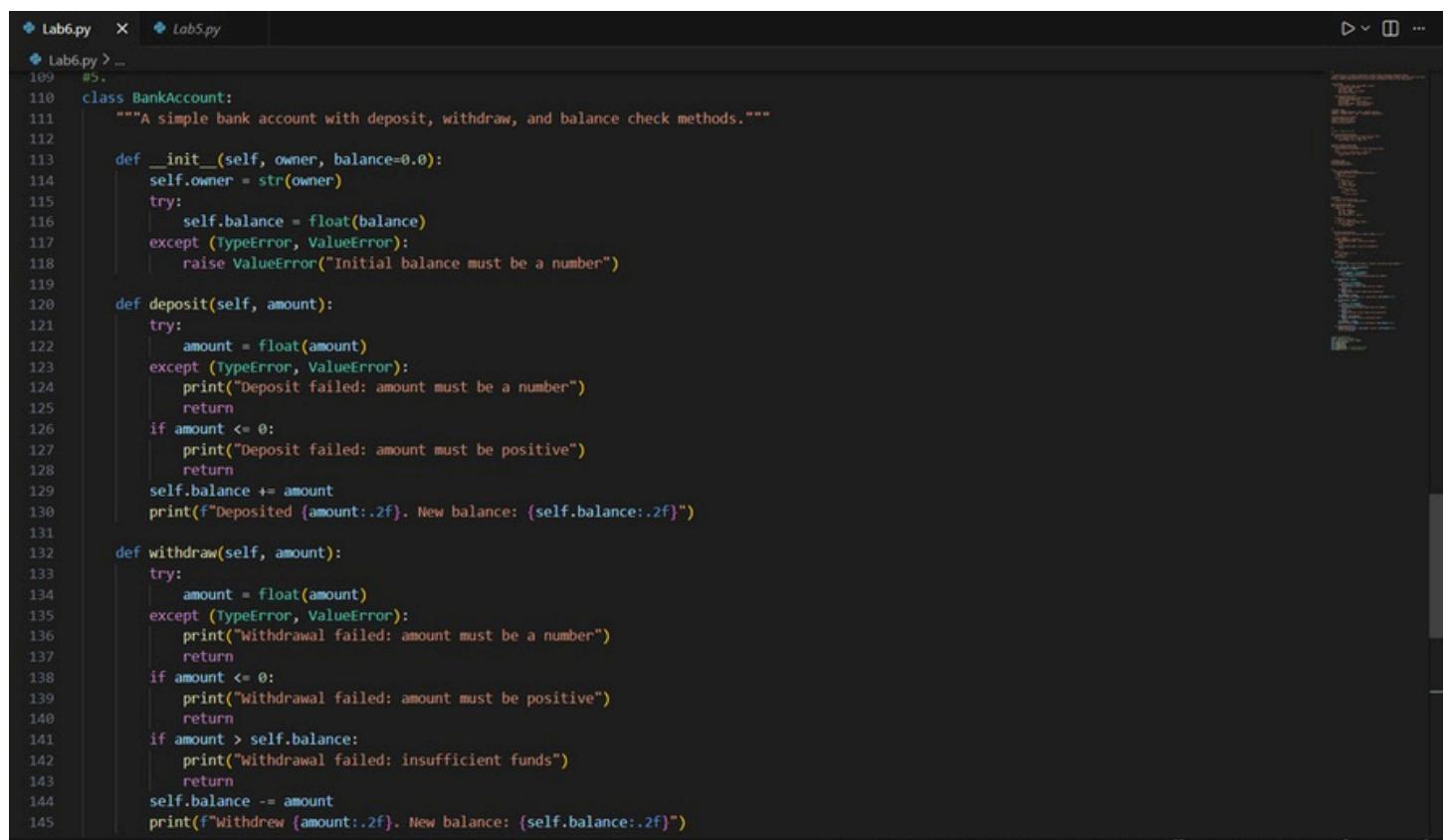
5.Scenario

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.

code & output:



The screenshot shows a code editor window with a dark theme. The file being edited is 'Lab6.py'. The code defines a 'BankAccount' class with methods for depositing and withdrawing funds, and checking the balance. The code includes comments explaining the logic and handling of errors like type conversion and negative values.

```
 109 #5.
110 class BankAccount:
111     """A simple bank account with deposit, withdraw, and balance check methods."""
112
113     def __init__(self, owner, balance=0.0):
114         self.owner = str(owner)
115         try:
116             self.balance = float(balance)
117         except (TypeError, ValueError):
118             raise ValueError("Initial balance must be a number")
119
120     def deposit(self, amount):
121         try:
122             amount = float(amount)
123         except (TypeError, ValueError):
124             print("Deposit failed: amount must be a number")
125             return
126         if amount <= 0:
127             print("Deposit failed: amount must be positive")
128             return
129         self.balance += amount
130         print(f"Deposited {amount:.2f}. New balance: {self.balance:.2f}")
131
132     def withdraw(self, amount):
133         try:
134             amount = float(amount)
135         except (TypeError, ValueError):
136             print("Withdrawal failed: amount must be a number")
137             return
138         if amount <= 0:
139             print("Withdrawal failed: amount must be positive")
140             return
141         if amount > self.balance:
142             print("Withdrawal failed: insufficient funds")
143             return
144         self.balance -= amount
145         print(f"Withdrew {amount:.2f}. New balance: {self.balance:.2f}")
```

```
Lab6.py  X  Lab5.py
* Lab6.py > ...
110  class BankAccount:
111      def withdraw(self, amount):
112          ...
113          if amount <= 0:
114              print("Withdrawal failed: amount must be positive")
115              return
116          if amount > self.balance:
117              print("Withdrawal failed: insufficient funds")
118              return
119          self.balance -= amount
120          print(f"Withdrew {amount:.2f}. New balance: {self.balance:.2f}")
121
122      def check_balance(self):
123          print(f"Account owner: {self.owner}, balance: {self.balance:.2f}")
124          return self.balance
125
126
127  # Demo for BankAccount
128  print('\nBankAccount demo:')
129  acct = BankAccount("Alice", 100.0)
130  acct.check_balance()
131  acct.deposit(50)
132  acct.withdraw(30)
133  acct.withdraw(200) # insufficient funds
134  acct.deposit("abc") # invalid deposit
```

```
PS C:\Users\Asus\OneDrive\Desktop\AIAC> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe
BankAccount demo:
Account owner: Alice, balance: 100.00
Deposited 50.00. New balance: 150.00
Withdrew 30.00. New balance: 120.00
Withdrawal failed: insufficient funds
Deposit failed: amount must be a number
PS C:\Users\Asus\OneDrive\Desktop\AIAC> [ ]
```

Prompt:

you are designing a banking application.create a Bank Account class with methods such as deposit(),withdraw(),and check_balance().