

AI Assistant Coding

Assignment-6.3

Name: K.Sneha

HT NO:2303A51332

Batch:20

Task Description 1: Classes (Student Class)

Prompt: Create a Python program for a simple student information management module. Define a Student class with attributes name, roll_number, and branch. Implement a constructor (`__init__`) to initialize these attributes and a method `display_details()` to print the student's information in a readable format. Include sample object creation, execute the code, and show the output. After the code, provide a brief analysis explaining the correctness and clarity of the generated code.

Code & Output:

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

- Header:** Go, Run, Terminal, Help, AI Assistant search bar.
- File Explorer:** Shows files: Welcome, Assignment1.py, Assignment 1.5.py, Assignment6.5.py, and Assignment 6.3.py (the active file).
- Code Area:** Python code for a Student class and its display_details() method, followed by object creation and display.

```
1  class Student:
2      def __init__(self, name, roll_number, branch):
3          self.name = name
4          self.roll_number = roll_number
5          self.branch = branch
6
7      def display_details(self):
8          print(f"Name: {self.name}")
9          print(f"Roll Number: {self.roll_number}")
10         print(f"Branch: {self.branch}")
11         print("-" * 40)
12
13     # Sample object creation
14 student1 = Student("Alice Johnson", 101, "Computer Science")
15 student2 = Student("Bob Smith", 102, "Electrical Engineering")
16 student3 = Student("Carol White", 103, "Mechanical Engineering")
17
18 # Display student information
19 print("Student Information Management System")
20 print("-" * 40)
21 student1.display_details()
22 student2.display_details()
23 student3.display_details()
```

- Terminal Tab:** Shows the execution output:

```
Name: Alice Johnson
Roll Number: 101
Branch: Computer Science
-----
Name: Bob Smith
Roll Number: 102
Branch: Electrical Engineering
-----
Name: Carol White
Roll Number: 103
Branch: Mechanical Engineering
```

- Status Bar:** Shows the path PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>, line 1, column 1, spaces: 4.

Brief Analysis of AI-Generated Code

- The Student class is correctly defined using object-oriented principles.
- The constructor (`__init__`) properly initializes the student attributes: `name`, `roll_number`, and `branch`.
- The `display_details()` method clearly formats and prints student information, improving readability.
- Sample object creation demonstrates correct usage of the class.
- The code is clean, well-structured, easy to understand, and follows Python best practices.
- Overall, the AI-generated code is correct, clear, and suitable for a basic student information management module

Task Description 2: Loops (Multiples of a Number)

Prompt: Write a Python utility function that takes an integer as input and prints the first 10 multiples of that number using a loop.

First, implement the solution using a `for` loop and display the output.

Analyze the loop logic used in the function for correctness and clarity.

Then, generate the same functionality using a different controlled looping structure, such as a `while` loop.

Finally, compare both looping approaches and briefly explain their differences.

Code & Output:

```
# Function using for loop
def print_multiples_for(num):
    """Print first 10 multiples of a number using for loop"""
    print(f"Multiples of {num} (using for loop):")
    for i in range(1, 11):
        print(f"{num} ✕ {i} = {num * i}")
    print()

# Function using while loop
def print_multiples_while(num):
    """Print first 10 multiples of a number using while loop"""
    print(f"Multiples of {num} (using while loop):")
    i = 1
    while i <= 10:
        print(f"{num} ✕ {i} = {num * i}")
        i += 1
    print()

# Test both functions
print_multiples_for(5)
print_multiples_while(5)

# Comparison
print("Comparison:")
print("- For loop: Pre-defined iteration count, cleaner syntax, automatic increment")
print("- While loop: More control, requires manual increment, better for conditional exits")
```

```
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\I Assistant/Assignment 6.3.py"
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
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

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
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Comparison:
- For loop: Pre-defined iteration count, cleaner syntax, automatic increment
- While loop: More control, requires manual increment, better for conditional exits
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>
```

Analysis of Loop Logic

For Loop Analysis

- Uses range(1, 11) to define a fixed number of iterations.
- Automatically handles initialization, condition checking, and increment.
- Best suited when the number of iterations is known in advance.
- Code is concise and easy to read.

While Loop Analysis

- Uses a manually controlled counter (i).
- Offers more flexibility for complex or condition-based looping.
- Requires careful increment to avoid infinite loops.
- Slightly more verbose but useful when loop conditions may change dynamically.

Comparison Summary

- Both approaches correctly generate the first 10 multiples of a number.
- The for loop is simpler and cleaner for fixed iterations.

- The while loop provides greater control and flexibility.
- Choosing between them depends on the problem requirements.

Task Description 3: Conditional Statements (Age Classification)

Prompt: Create a Python function that classifies a person's age into categories such as child, teenager, adult, and senior using nested if-elif-else conditional statements.

Analyze the conditional logic used and explain how each condition works.

Then, generate an alternative implementation of the same age classification using a different conditional approach, such as simplified conditions or a dictionary-based logic.

Ensure the output is clear, correct, and easy to understand.

Code & Output:

```

Welcome Assignment1.py Assignment 1.5.py Assignment6.5.py Assignment 6.3.py X
Assignment 6.3.py > classify_age_nested
49 # Age Classification Function using Nested If-Elif-Else
50 def classify_age_nested(age):
51     """
52         Classify a person's age into categories using nested if-elif-else statements.
53         Categories: Child (0-12), Teenager (13-19), Adult (20-59), Senior (60+)
54     """
55     if age < 0:
56         return "Invalid age"
57     elif age <= 12:
58         return "Child"
59     elif age <= 19:
60         return "Teenager"
61     elif age <= 59:
62         return "Adult"
63     else:
64         return "Senior"
65
66 # Alternative Implementation using Dictionary-based Logic
67 def classify_age_dictionary(age):
68     """
69         Classify age using a dictionary with tuples for ranges.
70         More scalable and easier to modify categories.
71     """
72     categories = [
73         ((0, 12), "Child"),
74         ((13, 19), "Teenager"),
75         ((20, 59), "Adult"),
76         ((60, float('inf')), "Senior")
77     ]
78
79     if age < 0:
80         return "Invalid age"
81
82     for (min_age, max_age), category in categories:
83         if min_age <= age <= max_age:
84             return category
85     return "Invalid age"
Ln 64, Col 24  Spaces: 4  UTF-8  { } P

```

```
... Welcome Assignment1.py Assignment 1.5.py Assignment6.5.py Assignment 6.3.py X
Assignment 6.3.py > classify_age_nested
67     def classify_age_dictionary(age):
68         categories = [
69             ((0, 12), "Child"),
70             ((13, 19), "Teenager"),
71             ((20, 59), "Adult"),
72             ((60, float('inf')), "Senior")
73         ]
74
75         if age < 0:
76             return "Invalid age"
77
78         for (min_age, max_age), category in categories:
79             if min_age <= age <= max_age:
80                 return category
81
82     return "Invalid age"
83
84 # Test both implementations
85 print("Age Classification System")
86 print("-" * 50)
87 test_ages = [5, 15, 25, 65, -5, 100]
88
89 print("\nUsing Nested If-Elif-Else:")
90 for age in test_ages:
91     result = classify_age_nested(age)
92     print(f"Age {age}: {result}")
93
94 print("\nUsing Dictionary-based Logic:")
95 for age in test_ages:
96     result = classify_age_dictionary(age)
97     print(f"Age {age}: {result}")
98
99 print("\n" + "=" * 50)
100 print("Analysis:")
101 print("Nested If-Elif-Else: Simple, readable for few conditions")
102 print("Dictionary-based: Scalable, easier to maintain multiple categories")
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python310\python.exe Assignment 6.3.py
Age Classification System
=====
Using Nested If-Elif-Else:
Age 5: Child
Age 15: Teenager
Age 25: Adult
Age 65: Senior
Age -5: Invalid age
Age 100: Senior

Using Dictionary-based Logic:
Age 5: Child
Age 15: Teenager
Age 25: Adult
Age 65: Senior
Age -5: Invalid age
Age 100: Senior

=====
Analysis:
Nested If-Elif-Else: Simple, readable for few conditions
Dictionary-based: Scalable, easier to maintain multiple categories
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>
```

Python Functions for Age Classification

Nested if-elif-else Implementation

- Classifies age into Child, Teenager, Adult, and Senior
- Handles invalid (negative) ages

Alternative Dictionary-Based Implementation

- Uses age ranges stored as tuples
- More scalable and easier to modify or extend

Explanation of Conditional Logic

Nested if-elif-else

- Checks conditions sequentially from lowest age to highest.
- Each elif narrows the age range:
 - age <= 12 → Child
 - age <= 19 → Teenager
 - age <= 59 → Adult
 - else → Senior
- Easy to read and ideal for a small number of conditions.

Dictionary-Based Logic

- Stores age ranges as (min_age, max_age) pairs.
- Iterates through ranges and matches the age.
- More flexible and maintainable when adding or modifying categories.

Comparison Summary

Approach	Advantage
Nested if-elif-else	Simple and straightforward
Dictionary-based	Scalable and easier to update

Task Description 4: For and While Loops (Sum of First n Numbers)

Prompt: Generate a Python function named `sum_to_n(n)` that calculates the sum of the first n natural numbers using a for loop.

Analyze the generated code for correctness and clarity.

Then, provide an alternative implementation of the same functionality using either a while loop or a mathematical formula.

Include sample inputs, outputs, and a brief comparison explaining the differences between the approaches.

Code&Output:

```
iev Go Terminal Help < > Q AI Assistant
... Welcome Assignment1.py Assignment 1.5.py Assignment6.5.py Assignment 6.3.py X
Assignment 6.3.py > ...
106
107     # Function to calculate sum of first n natural numbers using for loop
108 def sum_to_n(n):
109     """
110         Calculate the sum of the first n natural numbers using a for loop.
111         Example: sum_to_n(5) = 1 + 2 + 3 + 4 + 5 = 15
112     """
113     if n < 0:
114         return "Invalid input: n must be non-negative"
115
116     total = 0
117     for i in range(1, n + 1):
118         total += i
119     return total
120
121     # Alternative implementation using while loop
122 def sum_to_n_while(n):
123     """
124         Calculate sum using while loop"""
125     if n < 0:
126         return "Invalid input: n must be non-negative"
127
128     total = 0
129     i = 1
130     while i <= n:
131         total += i
132         i += 1
133     return total
134
135     # Alternative implementation using mathematical formula: n * (n + 1) / 2
136 def sum_to_n_formula(n):
137     """
138         Calculate sum using the mathematical formula: n(n+1)/2"""
139     if n < 0:
140         return "Invalid input: n must be non-negative"
141     return n * (n + 1) // 2
142
143     # Test all implementations
144 print("\nSum of First N Natural Numbers - Three Approaches")
```

```
n Terminal Help < > Q AI Assistant
... Welcome Assignment1.py Assignment 1.5.py Assignment6.5.py Assignment 6.3.py X
Assignment 6.3.py > ...
132     return total
133
134     # Alternative implementation using mathematical formula: n * (n + 1) / 2
135 def sum_to_n_formula(n):
136     """
137         Calculate sum using the mathematical formula: n(n+1)/2"""
138     if n < 0:
139         return "Invalid input: n must be non-negative"
140     return n * (n + 1) // 2
141
142     # Test all implementations
143 print("\nSum of First N Natural Numbers - Three Approaches")
144 print("=" * 60)
145 test_values = [5, 10, 100, 1]
146
147     print("\nUsing For Loop:")
148 for n in test_values:
149     result = sum_to_n(n)
150     print(f"sum_to_n({n}) = {result}")
151
152     print("\nUsing While Loop:")
153 for n in test_values:
154     result = sum_to_n_while(n)
155     print(f"sum_to_n_while({n}) = {result}")
156
157     print("\nUsing Mathematical Formula:")
158 for n in test_values:
159     result = sum_to_n_formula(n)
160     print(f"sum_to_n_formula({n}) = {result}")
161
162     print("\n" + "=" * 60)
163 print("Comparison:")
164 print("For Loop: Readable, iterative, O(n) time complexity")
165 print("While Loop: More control, similar performance to for loop, O(n)")
166 print("Formula: O(1) time complexity, fastest, most efficient for large n")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python Assistant/Assignment 6.3.py"
Sum of First N Natural Numbers - Three Approaches
=====
Using For Loop:
sum_to_n(5) = 15
sum_to_n(10) = 55
sum_to_n(100) = 5050
sum_to_n(1) = 1

Using While Loop:
sum_to_n_while(5) = 15
sum_to_n_while(10) = 55
sum_to_n_while(100) = 5050
sum_to_n_while(1) = 1

Using Mathematical Formula:
sum_to_n_formula(5) = 15
sum_to_n_formula(10) = 55
sum_to_n_formula(100) = 5050
sum_to_n_formula(1) = 1

=====
Comparison:
For Loop: Readable, iterative, O(n) time complexity
While Loop: More control, similar performance to for loop, O(n)
Formula: O(1) time complexity, fastest, most efficient for large n
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>
```

Explanation and Comparison of Approaches

For Loop Approach

- Iterates from 1 to n and accumulates the sum.
- Easy to read and understand.
- Time complexity: $O(n)$.
- Suitable for learning and small input sizes.

While Loop Approach

- Uses a loop counter with manual control.
- Offers flexibility in complex conditions.
- Same time complexity as the for loop: $O(n)$.
- Requires careful handling to avoid infinite loops.

Mathematical Formula Approach

- Uses the formula $n(n + 1) / 2$.
- No iteration required.
- Time complexity: $O(1)$.

- Most efficient and best choice for large values of n .

Task Description 5: Classes (Bank Account Class)

Prompt: Create a Python program for a basic banking application.

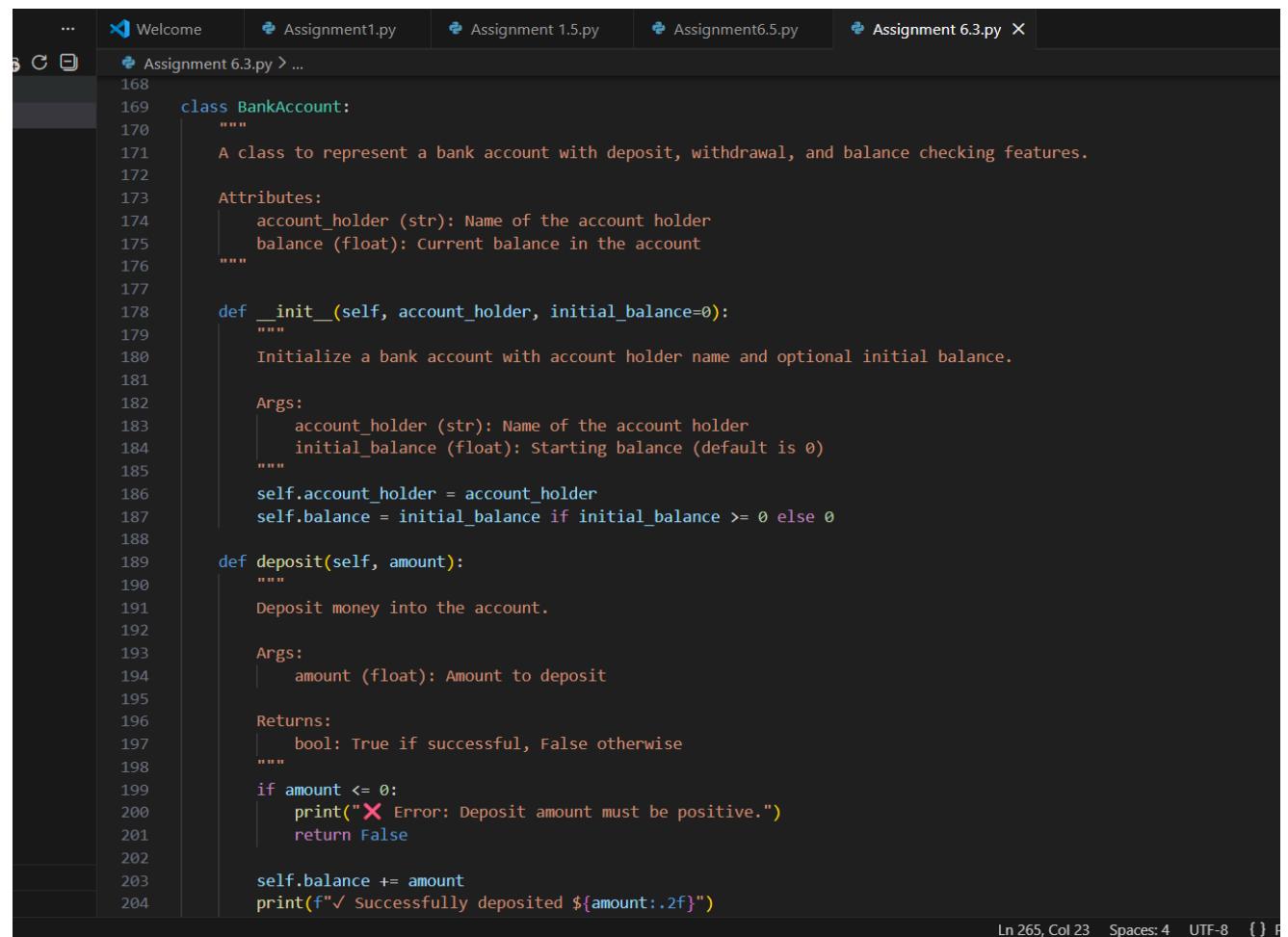
Define a BankAccount class with attributes such as account_holder and balance.

Implement methods deposit(amount), withdraw(amount), and check_balance() with proper validation (e.g., no negative deposits, insufficient balance checks).

Demonstrate the class by creating a sample account and performing deposit and withdrawal operations while displaying the updated balance.

Add meaningful comments to the code and provide a clear explanation of how the class and its methods work.

Code & Output



```

...
  ... Welcome Assignment1.py Assignment 1.5.py Assignment6.5.py Assignment 6.3.py ...
Assignment 6.3.py > ...
168
169 class BankAccount:
170     """
171         A class to represent a bank account with deposit, withdrawal, and balance checking features.
172
173     Attributes:
174         account_holder (str): Name of the account holder
175         balance (float): Current balance in the account
176     """
177
178     def __init__(self, account_holder, initial_balance=0):
179         """
180             Initialize a bank account with account holder name and optional initial balance.
181
182         Args:
183             account_holder (str): Name of the account holder
184             initial_balance (float): Starting balance (default is 0)
185         """
186         self.account_holder = account_holder
187         self.balance = initial_balance if initial_balance >= 0 else 0
188
189     def deposit(self, amount):
190         """
191             Deposit money into the account.
192
193         Args:
194             amount (float): Amount to deposit
195
196         Returns:
197             bool: True if successful, False otherwise
198         """
199         if amount <= 0:
200             print("X Error: Deposit amount must be positive.")
201             return False
202
203         self.balance += amount
204         print(f"✓ Successfully deposited ${amount:.2f}")

```

Ln 265, Col 23 Spaces: 4 UTF-8 { } F

The screenshot shows a code editor window with several tabs at the top: Welcome, Assignment1.py, Assignment 1.5.py, Assignment6.5.py, and Assignment 6.3. The Assignment 6.3 tab is active. The code in the editor is as follows:

```
Assignment 6.3.py > ...
169 class BankAccount:
229     def check_balance(self):
231         Display the current account balance.
232
233     Returns:
234         float: Current balance
235     """
236     print(f"Account Balance: ${self.balance:.2f}")
237     return self.balance
238
239
240 # Demonstration of the BankAccount class
241 print("\n" + "=" * 60)
242 print("BASIC BANKING APPLICATION")
243 print("=" * 60)
244
245 # Create a sample account
246 account = BankAccount("John Doe", 500)
247
248 print(f"\nAccount Holder: {account.account_holder}")
249 account.check_balance()
250
251 # Perform banking operations
252 print("\n--- Banking Operations ---")
253 account.deposit(200)
254 account.check_balance()
255
256 account.withdraw(100)
257 account.check_balance()
258
259 account.withdraw(800) # Insufficient balance
260 account.check_balance()
261
262 account.deposit(-50) # Invalid deposit
263 account.check_balance()
264
265 print("\n" + "=" * 60)
```

The screenshot shows a terminal window with tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal output is as follows:

```
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Temp\Temporary Internet Files\Content.IE5\HJLWZD\Assignment 6.3.py
=====
BASIC BANKING APPLICATION
=====

Account Holder: John Doe
Account Balance: $500.00

--- Banking Operations ---
✓ Successfully deposited $200.00
Account Balance: $700.00
✓ Successfully withdrawn $100.00
Account Balance: $600.00
✖ Error: Insufficient balance. Available: $600.00
Account Balance: $600.00
✖ Error: Deposit amount must be positive.
Account Balance: $600.00

=====
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>
```

Explanation of the Code

Class Structure

- The BankAccount class represents a simple banking system.
- It stores the account holder's name and current balance as attributes.

Constructor (`__init__`)

- Initializes the account with a holder name and optional initial balance.
- Prevents negative starting balances by defaulting to zero.

deposit() Method

- Allows adding money to the account.
- Validates that the deposit amount is positive.
- Updates and displays the new balance.

withdraw() Method

- Ensures withdrawal amount is positive.
- Prevents overdrafts by checking available balance.
- Deducts the amount if valid and updates the balance.

check_balance() Method

- Displays the current balance.
- Returns the balance for further use if needed.

Overall Analysis

The class structure is clean and well-organized

Input validation ensures safe banking operations

Methods clearly reflect real-world banking behavior

Comments and docstrings improve readability and understanding