

Assignment-6.3

Hall ticket no:2303A51523

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.

Prompt:develop a simple student information management module using python

In this task, an AI tool was used to generate a Python `Student` class with attributes such as name, roll number, and branch. The class included a constructor to initialize values and a `display_details()` method to print student information. The generated code was clear, correct, and followed basic object-oriented programming principles.

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar shows file navigation. The main code area contains the following Python code:

```

C:\> Users > appam > AppData > Local > Packages > 5319275A.What'sAppDesktop_cvlg1gavmjgm > LocalState > sessions > ADE69961C7978A860A3B495981C384FAD2CC1095 > transfers > 2026-05 > assignment 6.3.py > Student
1 # Lab 6: AI-Based Code Completion - Classes, Loops, and Conditionals
2
3 print("/*")
4 print("TASK 1: STUDENT CLASS")
5 print("*/")
6
7 # Task 1: Student Class
8 class Student:
9     """A class to represent a student with basic information."""
10
11     def __init__(self, name, roll_number, branch):
12         """
13             Constructor to initialize student attributes.
14
15             Args:
16                 name (str): Student's name
17                 roll number (int): Student's roll number
18                 branch (str): Student's branch/department
19
20             ...
21
22         self.name = name
23         self.roll_number = roll_number
24         self.branch = branch

```

Below the code, there is a PROBLEMS tab with one error, an OUTPUT tab, and a DEBUG CONSOLE tab. The DEBUG CONSOLE tab shows the following output:

```

=====
TASK 1: STUDENT CLASS
=====

Student 1:
Name: Alice Johnson
Roll Number: 101
Branch: Computer Science

Student 2:
Name: Anil Yadav
Roll Number: 102
Branch: Electronics

```

The status bar at the bottom shows the terminal type as powershell, Python version as 3.14.3, and the date/time as 04-02-2026.

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).

while instead of for).

Prompt:print first 10 multiples of a number

AI assistance was used to create a function that prints the first ten multiples of a given number using a `for` loop. The same logic was also generated using a `while`

loop. Both approaches produced correct results, with the `for` loop being simpler and more readable, while the `while` loop offered more control.

The screenshot shows a code editor window with a dark theme. The file is named `assignment 6.3.py`. The code contains two functions: `print_multiples_for` and `print_multiples_while`. Both functions print the first 10 multiples of a given number. The `for` loop version is simpler and cleaner, while the `while` loop version is more verbose. The output terminal shows the results for both loops, which are identical. A tooltip provides analysis of the loops: "FOR loop: Best for known iterations, cleaner syntax, more Pythonic". The system tray at the bottom shows the date as 04-02-2026.

```
C:\> Users > appam > AppData > Local > Packages > 5319275A.WhatAppDesktop_cv1g1gvanyjgm > LocalState > sessions > ADE69961C7978A860A3B495981C384FAD2CC1095 > transfers > 2026-05 > assignment 6.3.py > Student

File Edit Selection View Go Run Terminal Help ← → Search

assignment 6.3.py X
C:\> print(" - 70")
42
43 # Task 2: Loops - Print first 10 multiples of a number
44
45 # Approach 1: Using for loop
46 def print_multiples_for(number):
47     """Print first 10 multiples of a given number using for loop."""
48     print(f"\nFirst 10 multiples of {number} (using for loop):")
49     for i in range(1, 11):
50         multiple = number * i
51         print(f"\t{number} x {i} = {multiple}", end=" ")
52     print()
53
54 # Approach 2: Using while loop
55 def print_multiples_while(number):
56     """Print first 10 multiples of a given number using while loop."""
57     print(f"\nFirst 10 multiples of {number} (using while loop):")
58     i = 1
59     while i <= 10:
60         multiple = number * i
61         print(f"\t{number} x {i} = {multiple}", end=" ")
62         i += 1
63     print()

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

+ ... | x
powershell
Python Deb...
Ln 10, Col 5 Spaces: 4 UTF-8 CRLF Python 3.14.3 09:56
ENG IN 04-02-2026

TASK 2: LOOPS (MULTIPLES OF A NUMBER)
-----
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 5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5 x 9 = 45 5 x 10 = 50

First 10 multiples of 7 (using while loop):
7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 7 x 4 = 28 7 x 5 = 35 7 x 6 = 42 7 x 7 = 49 7 x 8 = 56 7 x 9 = 63 7 x 10 = 70

Analysis of Looping Approaches:
- FOR loop: Best for known iterations, cleaner syntax, more Pythonic

68°F Sunny
Search
Ln 10, Col 5 Spaces: 4 UTF-8 CRLF Python 3.14.3 09:56
ENG IN 04-02-2026
```

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).

Prompt:conditional statements and age qualification using nested if-elif-else

explanation:The AI generated a function using `if-elif-else` statements to classify age into child, teenager, adult, and senior categories. The conditions were logically ordered and easy to understand. An alternative approach showed how the same classification could be done using simplified or dictionary-based logic.

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains code for nested if-elif-else statements:

```
C:\> Users>appnam>AppData>Local>Packages>5319275AWhatsAppDesktop_cvg1gvanyjgm>LocalStorage>sessions>ADE69961C7978A860A3B495981C384FAD2CC1095>transfers>2026-05>assignment 6.3.py>Student
75 print("=>76)
76
77 # Task 3: Conditional Statements - Age Classification
78
79 # Approach 1: Using nested if-elif-else
80 def classify_age_nested(age):
81     """Classify age into groups using nested if-elif-else statements."""
82     if age < 0:
83         return "Invalid age"
84     elif age < 13:
85         return "Child"
86     elif age < 18:
87         return "Teenager"
88     elif age < 60:
89         return "Adult"
90     else:
91         return "Senior"
92
93 # Approach 2: Using dictionary-based logic
94 def classify_age_dict(age):
95     """Classify age into groups using dictionary and conditional checks."""
96     age_groups = {
97         0: "Child",
98         13: "Teenager",
99         25: "Adult",
100        65: "Senior"
101    }
102
103    for key in age_groups:
104        if age <= key:
105            return age_groups[key]
106
107    return "Invalid age"
```

The second cell contains the output of the code, showing the classification of various ages:

```
TASK 3: CONDITIONAL STATEMENTS (AGE CLASSIFICATION)

Age Classification (Nested if-elif-else):
Age 5: Child
Age 15: Teenager
Age 25: Adult
Age 65: Senior

Age Classification (Dictionary-based):
Age 5: Child
```

At the bottom, the status bar shows: Ln 10, Col 5, Spaces: 4, UTF-8, CRLF, Python, 3.14.3, 1001, ENG IN, 04-02-2026.

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

t

task4.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.

Formula.

Prompt:give first sum of n natural numbers using for loop and while loop

explanation: AI was used to generate a function that calculates the sum of the first n natural numbers using a `for` loop. Alternative solutions using a `while` loop and a mathematical formula were also suggested. The formula-based approach was the most efficient, while loop-based solutions were easier for beginners.

task5: 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

Prompt:create bank account class and demonstrate

The screenshot shows a code editor window with two tabs: 'Assignment.7.ipynb' and 'assignment 6.3.py'. The 'assignment 6.3.py' tab contains the following Python code:

```
C:\Users\appam>AppData>Local>Packages>5319275A\WhatsAppDesktop_cv1g1gvanyjgm>LocalState>sessions>ADE69961C7978A860A3B495981C384FAD2CC1095>transfers>2026-05>assignment 6.3.py > Student

170
171 # Task 5: Bank Account Class
172 class BankAccount:
173     """A class to represent a bank account with deposit and withdrawal operations."""
174
175     def __init__(self, account_holder, initial_balance=0):
176         """
177             Constructor to initialize a bank account.
178
179             Args:
180                 account_holder (str): Name of the account holder
181                 initial_balance (float): Initial balance in the account (default: 0)
182
183             self.account_holder = account_holder
184             self.balance = initial_balance
185             self.transaction_history = []
186
187         if initial_balance > 0:
188             self.transaction_history.append(f"Initial deposit: ${initial_balance:.2f}")
189
190     def deposit(self, amount):
191         """
192             Adds the specified amount to the current balance.
193
194             Args:
195                 amount (float): Amount to be deposited
196
197             self.balance += amount
198             self.transaction_history.append(f"Deposited ${amount:.2f}")
199
200     def withdraw(self, amount):
201         """
202             Subtracts the specified amount from the current balance.
203
204             Args:
205                 amount (float): Amount to be withdrawn
206
207             if amount > self.balance:
208                 raise ValueError("Insufficient balance")
209             self.balance -= amount
210             self.transaction_history.append(f"Withdrew ${amount:.2f}")
211
212     def check_balance(self):
213         """
214             Returns the current balance of the account.
215
216             Returns:
217                 float: Current balance
218
219             print(f"Current Balance: ${self.balance:.2f}")
220
221     def display_transaction_history(self):
222         """
223             Prints the transaction history of the account.
224
225             self.transaction_history
226
227             for transaction in self.transaction_history:
228                 print(transaction)
229
230             print("\nTransaction History:")
231             for transaction in self.transaction_history:
232                 print("- " + transaction)
233
234             print("\nAccount Holder: " + self.account_holder)
235             print("Current Balance: $" + str(self.balance))
236
237             print("\nCompleted - All Tasks Executed Successfully!")
```

The terminal output below the code shows the creation of a bank account for John Smith with an initial balance of \$1000, followed by a series of banking operations including deposits and withdrawals, and finally displays the transaction history and current balance.

```
TASK 5: BANK ACCOUNT CLASS
=====
Creating Bank Account for John Smith with initial balance $1000
Performing Banking Operations:
-----
✓ Deposited $500.00
✓ withdrew $200.00
✓ Deposited $150.00
Current Balance: $1450.00
✓ withdrew $600.00
Current Balance: $850.00
```

The screenshot shows a code editor window with two tabs: 'Assignment.7.ipynb' and 'assignment 6.3.py'. The 'assignment 6.3.py' tab contains the following Python code:

```
C:\Users\appam>AppData>Local>Packages>5319275A\WhatsAppDesktop_cv1g1gvanyjgm>LocalState>sessions>ADE69961C7978A860A3B495981C384FAD2CC1095>transfers>2026-05>assignment 6.3.py > Student

257     print("\nPerforming Banking Operations:")
258     print("." * 50)
259
260     account.deposit(500)
261     account.withdraw(200)
262     account.deposit(150)
263     account.check_balance()
264     account.withdraw(600)
265     account.check_balance()
266
267     # Try invalid operations
268     print("\nAttempting invalid operations:")
269     account.withdraw(10000) # Insufficient balance
270     account.deposit(-100) # Invalid amount
271
272     # Display final account details
273     account.display_account_details()
274
275     print("*" * 70)
276     print("TAB 6 COMPLETED - ALL TASKS EXECUTED SUCCESSFULLY!")
277     print("*" * 70)
```

The terminal output below the code shows the execution of the script, which performs various banking operations and handles invalid attempts, finally displaying the completed status.

```
=====
Account Holder: John Smith
Current Balance: $850.00
=====
Transaction History:
  • Initial deposit: $1000.00
  • Deposit: $500.00
  • Withdrawal: $200.00
  • Deposit: $150.00
  • Withdrawal: $600.00
=====
Completed - All Tasks Executed Successfully!
```

