

AI ASSISTED CODING

ASSIGNMENT-6.1

HTNO: 2303A510B2

NAME: K AKHIL

BATCH-14

Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based program.

PROMPT: Generate Python code to print all even numbers between 1 and N using a loop.

CODE AND OUTPUT:

```
6.1(evenNumbers).py > ...
1  #Generate Python code to print all even numbers between 1 and N using a loop.
2  def print_even_numbers(N):
3      # Loop through numbers from 1 to N
4      for number in range(1, N + 1):
5          # Check if the number is even
6          if number % 2 == 0:
7              print(number)
8  # Example usage
9  N = 20
10 print_even_numbers(N)

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/venNumbers.py"
2
4
6
8
10
12
14
16
18
20
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task demonstrates the effective use of an AI tool to generate a loop-based program. The AI correctly identified that a for loop is suitable for iterating through a fixed range of numbers. Conditional logic was applied to check even numbers using the modulo operator. This reduces manual effort and ensures logical accuracy while reinforcing basic looping concepts.

Task Description #2 (AI-Based Code Completion for Loop with

Conditionals)

Task: Use an AI code completion tool to combine loops and conditionals.

PROMPT: Generate Python code to count how many numbers in a list are even and odd.

CODE AND OUTPUT:

```
6.1(evenOdd).py > ...
1  #Generate Python code to count how many numbers in a list are even and odd.give code in user input
2  def count_even_odd(numbers):
3      even_count = 0
4      odd_count = 0
5
6      for number in numbers:
7          if number % 2 == 0:
8              even_count += 1
9          else:
10             odd_count += 1
11
12     return even_count, odd_count
13 # Example usage
14 user_input = input("Enter a list of numbers separated by spaces: ")
15 numbers = list(map(int, user_input.split()))
16 even_count, odd_count = count_even_odd(numbers)
17 print(f"Even numbers count: {even_count}")
18 print(f"Odd numbers count: {odd_count}")

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/AI(vs)/6.1(evenodd).py"
Enter a list of numbers separated by spaces: 2 54 6 33
Even numbers count: 3
Odd numbers count: 1
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task validates the AI's ability to combine loops and conditional statements effectively. The AI-generated solution correctly iterates through a list and classifies numbers as even or odd. The logic improves code clarity and efficiency, showing how AI can assist in writing structured programs involving decision-making and iteration.

Task Description #3 (AI-Based Code Completion for Class)

Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input.

PROMPT: Generate a Python class User that validates age and email using conditional statements.

CODE AND OUTPUT:

```
6.1(ageEmail).py > ...
1  #Generate a Python class User that validates age and email using conditional statements.give code in user input
2  class User:
3      def __init__(self, name, age, email):
4          self.name = name
5          self.age = age
6          self.email = email
7          self.validate_age()
8          self.validate_email()
9
10     def validate_age(self):
11         if not (0 < self.age < 120):
12             raise ValueError("Age must be between 1 and 119.")
13
14     def validate_email(self):
15         if "@" not in self.email or "." not in self.email.split("@")[-1]:
16             raise ValueError("Invalid email format.")
17
18     def get_user_info(self):
19         return {
20             "name": self.name,
21             "age": self.age,
22             "email": self.email
23         }
24 # Example usage
25 user_input_name = input("Enter your name: ")
26 user_input_age = int(input("Enter your age: "))
27 user_input_email = input("Enter your email: ")
28 user = User(user_input_name, user_input_age, user_input_email)
29 print(user.get_user_info())

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE

PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/ageEmail.py"
Enter your name: akhil
Enter your age: 20
Enter your email: akhil123@gmail.com
{'name': 'akhil', 'age': 20, 'email': 'akhil123@gmail.com'}
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task highlights the use of AI in generating class-based validation logic. The AI correctly implemented conditional checks for age and email format. This improves data reliability and demonstrates object-oriented programming concepts. AI assistance reduces development time while maintaining proper input validation mechanisms.

Task Description #4 (AI-Based Code Completion for Classes)

Task: Use an AI code completion tool to generate a Python class for managing student details.

PROMPT: Generate a Python class Student with attributes (name, roll number, marks) and methods to calculate total and average marks.

CODE :

```
❶ 6.1(class).py > ...
1  #: Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.
2  class Student:
3      def __init__(self, name, roll_number, marks):
4          self.name = name
5          self.roll_number = roll_number
6          self.marks = marks # marks should be a list of numbers
7
8      def calculate_total(self):
9          return sum(self.marks)
10
11     def calculate_average(self):
12         if len(self.marks) == 0:
13             return 0
14         return self.calculate_total() / len(self.marks)
15
16     def get_student_info(self):
17         return {
18             "name": self.name,
19             "roll_number": self.roll_number,
20             "marks": self.marks,
21             "total_marks": self.calculate_total(),
22             "average_marks": self.calculate_average()
23         }
24
25 # Example usage
26 student_name = input("Enter student name: ")
27 student_roll_number = input("Enter roll number: ")
28 marks_input = input("Enter marks separated by spaces: ")
29 marks_list = list(map(float, marks_input.split()))
30 student = Student(student_name, student_roll_number, marks_list)
31 print(student.get_student_info())
```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/D
Enter student name: akhil
Enter roll number: 10b2
Enter marks separated by spaces: 90 95 80 96
{'name': 'akhil', 'roll_number': '10b2', 'marks': [90.0, 95.0, 80.0, 96.0], 'total_marks': 361.0, 'average_marks': 90.25}
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task shows how AI helps in designing a **complete and well-structured class** with attributes and methods. The generated class correctly calculates total and average marks. Minor manual improvements were needed to handle edge cases, showing that AI provides a strong foundation while human oversight ensures robustness.

Task Description 5 (AI-Assisted Code Completion Review)

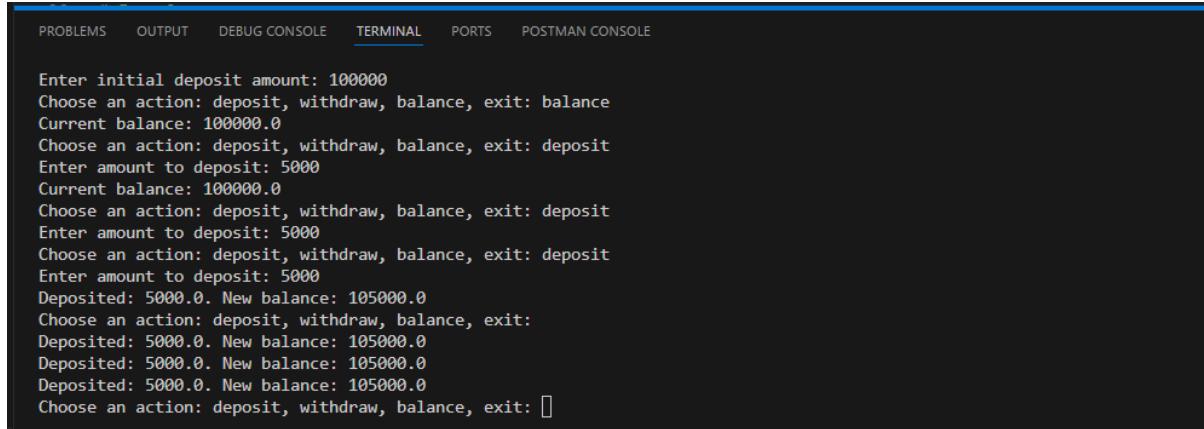
Task: Use an AI tool to generate a complete Python program using **classes, loops, and conditionals together.**

PROMPT: Generate a Python program for a simple bank account system using class, loops, and conditional statements.

CODE :

```
❶ 6.1(bank).py > ...
1  #Generate a Python program for a simple bank account system using class, loops, and conditional statements.
2  class BankAccount:
3      def __init__(self, account_holder, initial_balance=0):
4          self.account_holder = account_holder
5          self.balance = initial_balance
6
7      def deposit(self, amount):
8          if amount > 0:
9              self.balance += amount
10             print(f"Deposited: {amount}. New balance: {self.balance}")
11         else:
12             print("Deposit amount must be positive.")
13
14     def withdraw(self, amount):
15         if amount > 0:
16             if amount <= self.balance:
17                 self.balance -= amount
18                 print(f"Withdrew: {amount}. New balance: {self.balance}")
19             else:
20                 print("Insufficient funds for this withdrawal.")
21         else:
22             print("Withdrawal amount must be positive.")
23
24     def get_balance(self):
25         return self.balance
26
27 # Example usage
28 account_holder_name = input("Enter account holder name: ")
29 initial_deposit = float(input("Enter initial deposit amount: "))
30 account = BankAccount(account_holder_name, initial_deposit)
31 while True:
32     action = input("Choose an action: deposit, withdraw, balance, exit: ").lower()
33     if action == "deposit":
34         amount = float(input("Enter amount to deposit: "))
35         account.deposit(amount)
36     elif action == "withdraw":
37         amount = float(input("Enter amount to withdraw: "))
38         account.withdraw(amount)
39     elif action == "balance":
40         print(f"Current balance: {account.get_balance()}")
41     elif action == "exit":
42         print("Exiting the banking system. Goodbye!")
43         break
44     else:
45         print("Invalid action. Please choose again.")
```

OUTPUT:



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are tabs labeled PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and POSTMAN CONSOLE. The TERMINAL tab is currently selected. Below the tabs, the terminal displays the following text:

```
Enter initial deposit amount: 100000
Choose an action: deposit, withdraw, balance, exit: balance
Current balance: 100000.0
Choose an action: deposit, withdraw, balance, exit: deposit
Enter amount to deposit: 5000
Current balance: 100000.0
Choose an action: deposit, withdraw, balance, exit: deposit
Enter amount to deposit: 5000
Choose an action: deposit, withdraw, balance, exit: deposit
Enter amount to deposit: 5000
Deposited: 5000.0. New balance: 105000.0
Choose an action: deposit, withdraw, balance, exit:
Deposited: 5000.0. New balance: 105000.0
Deposited: 5000.0. New balance: 105000.0
Deposited: 5000.0. New balance: 105000.0
Choose an action: deposit, withdraw, balance, exit: []
```

JUSTIFICATION:

This task demonstrates AI's capability to generate a **complete working program** using classes, loops, and conditionals together. The AI-generated bank system simplifies program structure and improves coding speed. However, limitations such as missing validations highlight the importance of human judgment. Overall, AI significantly enhances coding productivity and learning efficiency.