

2303A510A1 - Bhanu

Batch - 14

Assignment - 6.1

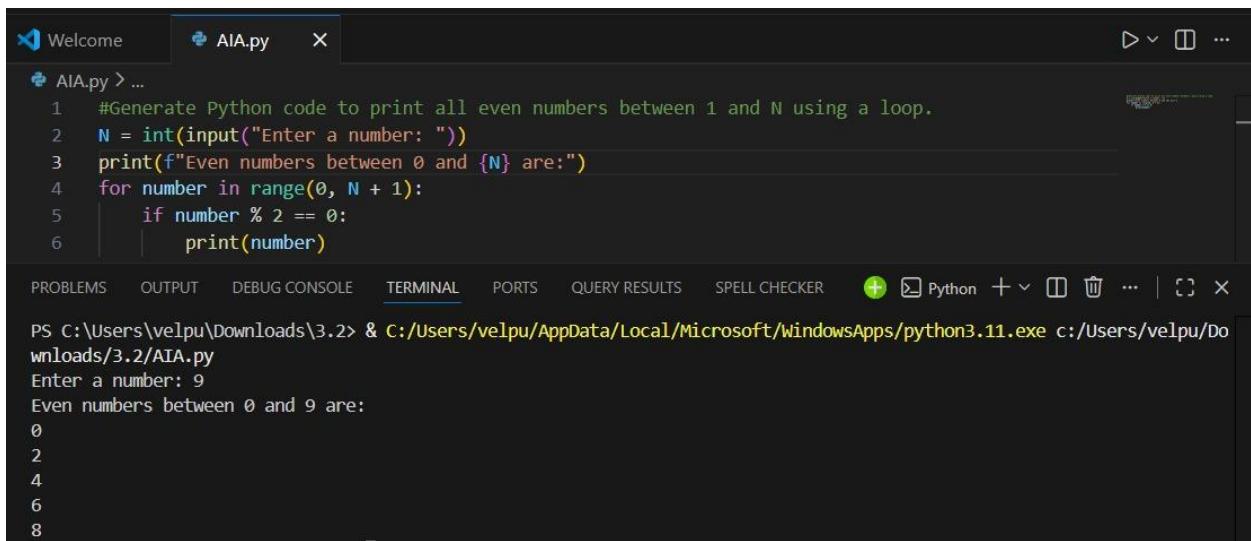
Experiment 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals

Task 1 : 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.

Program and Output:



The screenshot shows a code editor interface with a dark theme. A tab bar at the top has 'Welcome' and 'AI.py' selected. Below the tabs is a code editor window containing the following Python script:

```
#Generate Python code to print all even numbers between 1 and N using a loop.
N = int(input("Enter a number: "))
print(f"Even numbers between 0 and {N} are:")
for number in range(0, N + 1):
    if number % 2 == 0:
        print(number)
```

Below the code editor is a terminal window showing the execution of the script. The terminal output is:

```
PS C:\Users\velpu\Downloads\3.2> & C:/Users/velpu/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/velpu/Downloads/3.2/AIA.py
Enter a number: 9
Even numbers between 0 and 9 are:
0
2
4
6
8
```

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.

Program and Output:

The screenshot shows a code editor window with a dark theme. The file tab at the top says "AIA.py". The code in the editor is as follows:

```
1 #Generate Python code to count how many numbers in a list are even and odd.
2 def count_even_odd(numbers):
3     even_count = 0
4     odd_count = 0
5     for number in numbers:
6         if number % 2 == 0:
7             even_count += 1
8         else:
9             odd_count += 1
10    return even_count, odd_count
11 numbers = list(map(int, input("Enter numbers: ").split()))
12 even_count, odd_count = count_even_odd(numbers)
13 print(f"Count of Even numbers: {even_count}")
14 print(f"Count of Odd numbers: {odd_count}")
```

Below the code, the terminal tab is active, showing the command line interface output:

```
PS C:\Users\velpu\Downloads\3.2> & C:/Users/velpu/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/velpu/Downloads/3.2/AIA.py
Enter numbers: 1 2 3 4 5 6 7 8 9
Count of Even numbers: 4
Count of Odd numbers: 5
```

Explanation of logic flow:

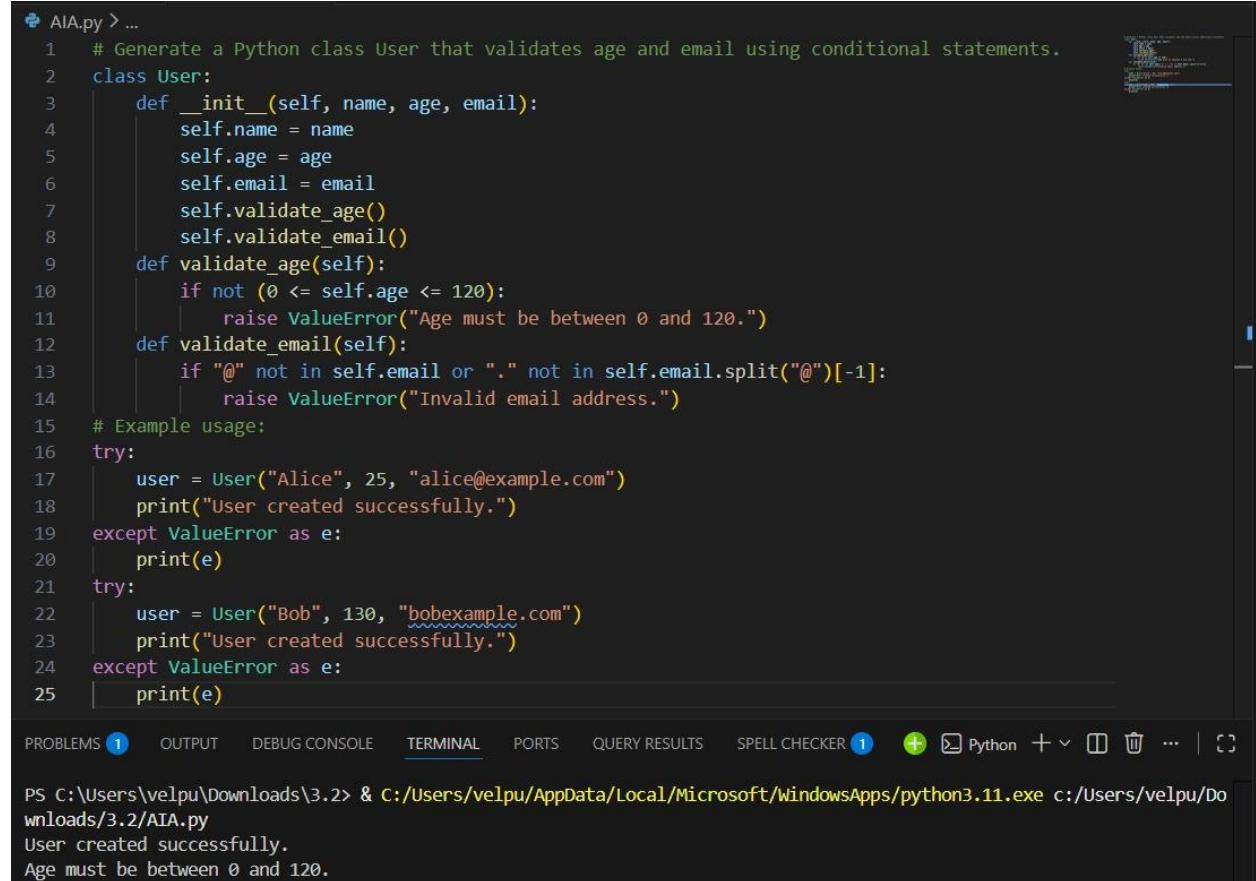
- This code defines a function 'count_even_odd' that takes a list of numbers as input.
- It initializes two counters, 'even_count' and 'odd_count', to zero.
- It then iterates through each number in the list, checking if it is even or odd using the modulus operator (%).
- If the number is even (i.e., divisible by 2), it increments the 'even_count' by 1; otherwise, it increments the 'odd_count' by 1.
- Finally, the function returns the counts of even and odd numbers.
- The user is prompted to enter a list of numbers, which are then converted to integers and stored in the 'numbers' list.
- The function is called with this list, and the results are printed to the console.

Task 3: 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.

Program and Output:



```
AI.py > ...
1 # Generate a Python class User that validates age and email using conditional statements.
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     def validate_age(self):
10        if not (0 <= self.age <= 120):
11            raise ValueError("Age must be between 0 and 120.")
12     def validate_email(self):
13        if "@" not in self.email or "." not in self.email.split("@")[-1]:
14            raise ValueError("Invalid email address.")
15 # Example usage:
16 try:
17     user = User("Alice", 25, "alice@example.com")
18     print("User created successfully.")
19 except ValueError as e:
20     print(e)
21 try:
22     user = User("Bob", 130, "bobexample.com")
23     print("User created successfully.")
24 except ValueError as e:
25     print(e)

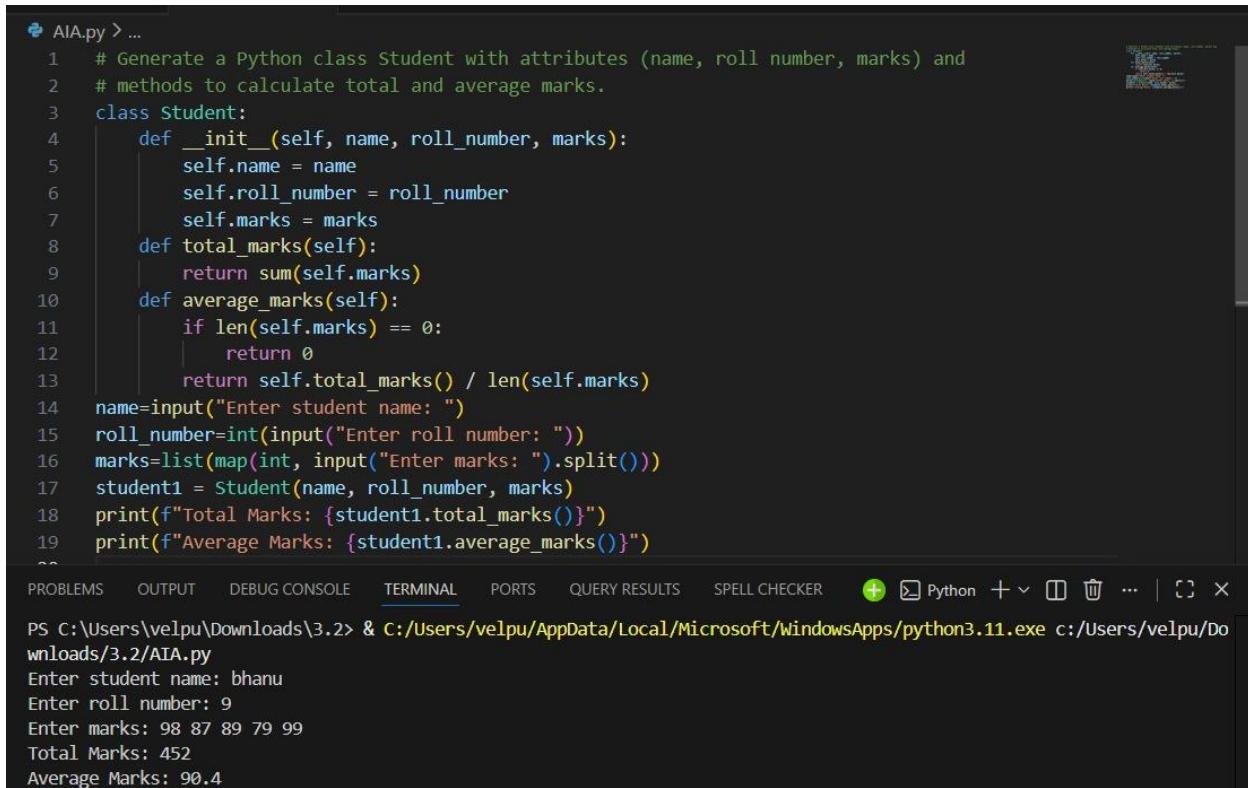
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS SPELL CHECKER 1 + Python + ... | ☰
PS C:\Users\velpu\Downloads\3.2> & C:/Users/velpu/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/velpu/Downloads/3.2/AI.py
User created successfully.
Age must be between 0 and 120.
```

Task 4: 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.

Program and Output:(Manual Improvement)



```
AIA.py > ...
1  # Generate a Python class Student with attributes (name, roll number, marks) and
2  # methods to calculate total and average marks.
3  class Student:
4      def __init__(self, name, roll_number, marks):
5          self.name = name
6          self.roll_number = roll_number
7          self.marks = marks
8      def total_marks(self):
9          return sum(self.marks)
10     def average_marks(self):
11         if len(self.marks) == 0:
12             return 0
13         return self.total_marks() / len(self.marks)
14 name=input("Enter student name: ")
15 roll_number=int(input("Enter roll number: "))
16 marks=list(map(int, input("Enter marks: ").split()))
17 student1 = Student(name, roll_number, marks)
18 print(f"Total Marks: {student1.total_marks()}")
19 print(f"Average Marks: {student1.average_marks()}")
...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS SPELL CHECKER + Python + ...
```

PS C:\Users\velpu\Downloads\3.2> & C:/Users/velpu/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/velpu/Downloads/3.2/AIA.py
Enter student name: bhanu
Enter roll number: 9
Enter marks: 98 87 89 79 99
Total Marks: 452
Average Marks: 90.4

Justification:

- The manual improvement adds a conditional check in the `average_marks()` method to handle the case where the marks list is empty.
- Without this check, the original AI-generated code may cause a runtime error (division by zero) if no marks are entered.
- By returning 0 when the marks list is empty, the improved version makes the program more robust, error-free, and reliable.
- This modification reflects responsible use of AI-generated code by reviewing, testing, and improving it to handle edge cases safely.

Program and Output:(AI generated)

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

```
1 # Generate a Python class Student with attributes (name, roll number, marks) and
2 # methods to calculate total and average marks.
3 class Student:
4     def __init__(self, name, roll_number, marks):
5         self.name = name
6         self.roll_number = roll_number
7         self.marks = marks
8     def total_marks(self):
9         return sum(self.marks)
10    def average_marks(self):
11        return self.total_marks() / len(self.marks)
12 name=input("Enter student name: ")
13 roll_number=int(input("Enter roll number: "))
14 marks=list(map(int, input("Enter marks: ").split()))
15 student1 = Student(name, roll_number, marks)
16 print(f"Total Marks: {student1.total_marks()}")
17 print(f"Average Marks: {student1.average_marks()}")
```

The terminal window below shows the execution of the script and its output:

```
PS C:\Users\velpu\Downloads\3.2> & C:/Users/velpu/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/velpu/Downloads/3.2/AIA.py
Enter student name: bhanu
Enter roll number: 9
Enter marks: 98 87 89 79 99
Total Marks: 452
Average Marks: 90.4
```

Justification:

- The AI-generated code correctly defines a Student class with appropriate attributes (name, roll_number, marks) and methods to calculate total marks and average marks.
- It efficiently uses built-in Python functions like sum() and len() to perform calculations, making the code simple, readable, and easy to understand.
- The logic works correctly when valid input (non-empty marks list) is provided, demonstrating proper use of classes, loops (implicit in sum()), and object-oriented programming concepts.

Conclusion

The AI-generated code provided a correct base implementation, while the manual improvement enhanced its stability and correctness, showing the importance of human oversight in AI-assisted coding.

Task 5: 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.

Program :

```

❸ AIAPy > -
1  # Generate a Python program for a simple bank account system using class, loops,
2  # and conditional statements.
3  class BankAccount:
4      def __init__(self, account_holder, initial_balance=0):
5          self.account_holder = account_holder
6          self.balance = initial_balance
7      def deposit(self, amount):
8          if amount > 0:
9              self.balance += amount
10             print(f"Deposited: ${amount:.2f}")
11         else:
12             print("Deposit amount must be positive.")
13     def withdraw(self, amount):
14         if amount > 0:
15             if amount <= self.balance:
16                 self.balance -= amount
17                 print(f"Withdraw: ${amount:.2f}")
18             else:
19                 print("Insufficient funds.")
20         else:
21             print("Withdrawal amount must be positive.")
22     def get_balance(self):
23         return self.balance
24 def main():
25     print("Welcome to the Simple Bank Account System")
26     account_holder = input("Enter account holder name: ")
27     account = BankAccount(account_holder)
28     while True:
29         print("\nMenu:")
30         print("1. Deposit")
31         print("2. Withdraw")
32         print("3. Check Balance")
33         print("4. Exit")
34         choice = input("Choose an option (1-4): ")
35         if choice == '1':
36             amount = float(input("Enter amount to deposit: "))
37             account.deposit(amount)
38         elif choice == '2':
39             amount = float(input("Enter amount to withdraw: "))
40             account.withdraw(amount)
41         elif choice == '3':
42             balance = account.get_balance()
43             print(f"Current balance: ${balance:.2f}")
44         elif choice == '4':
45             print("Thank you for using the Simple Bank Account System. Goodbye!")
46             break
47         else:
48             print("Invalid choice. Please try again.")
49     if __name__ == "__main__":
50         main()

```

Output:

```
Welcome to the Simple Bank Account System
Enter account holder name: sai pallavi

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 1
Enter amount to deposit: 1000
Deposited: $1000.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 2
Enter amount to withdraw: 200
Withdraw: $200.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance

Menu:
1. Deposit
2. Withdraw
3. Check Balance
Menu:
1. Deposit
2. Withdraw
3. Check Balance
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 3
Current balance: $800.00

Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 4
Thank you for using the Simple Bank Account System. Goodbye!
```