

AI ASSISTED CODING

Lab Assignment-6.1

Name : T. Shylasri

H.T.NO: 2303A51876

Batch-14(LAB-6)

Date: 02-02-2026

ASSIGNMENT-6.1

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.”

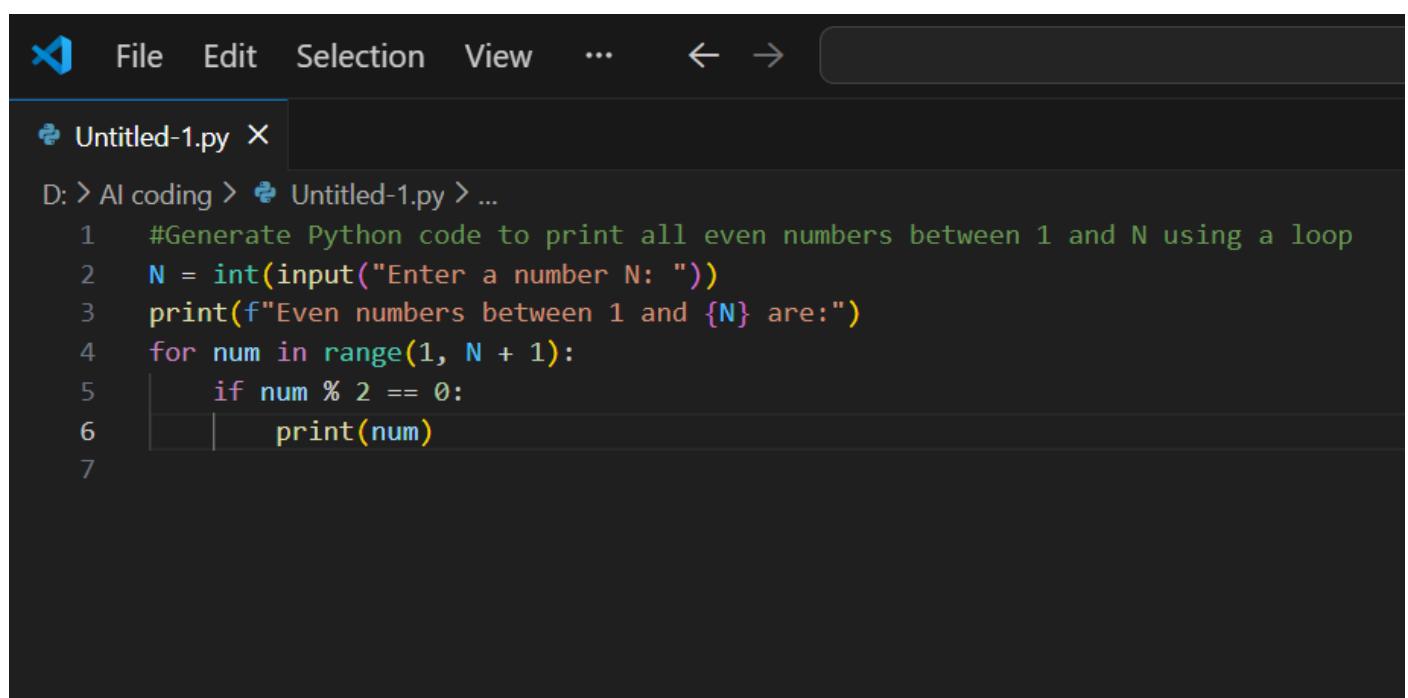
Expected Output:

- AI-generated loop logic.
- Identification of loop type used (for or while).
- Validation with sample inputs.

Prompt:

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

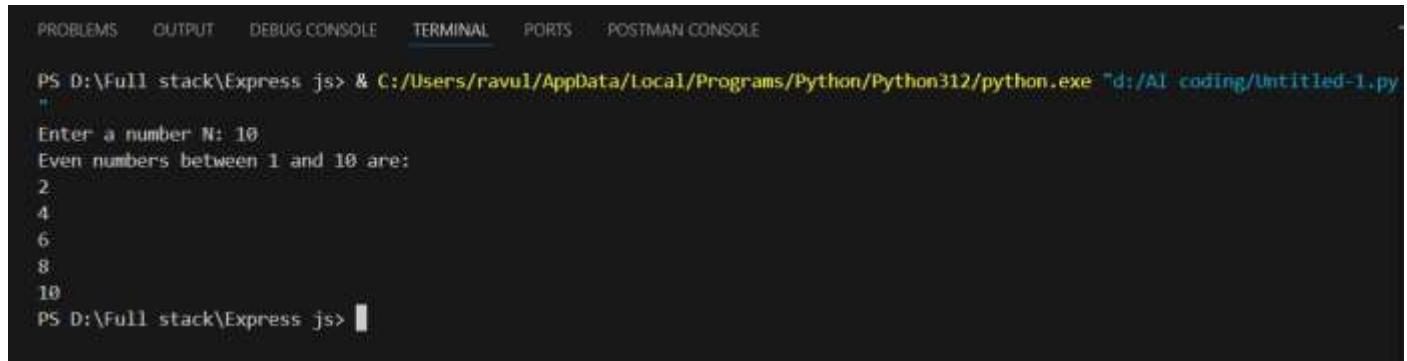
Code:



The screenshot shows a code editor interface with a dark theme. At the top, there's a menu bar with File, Edit, Selection, View, and other options. Below the menu is a tab bar showing 'Untitled-1.py'. The main area displays the following Python code:

```
D: > AI coding > Untitled-1.py > ...
1  #Generate Python code to print all even numbers between 1 and N using a loop
2  N = int(input("Enter a number N: "))
3  print(f"Even numbers between 1 and {N} are:")
4  for num in range(1, N + 1):
5      if num % 2 == 0:
6          print(num)
7
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS D:\Full stack\Express js> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/Untitled-1.py"
Enter a number N: 10
Even numbers between 1 and 10 are:
2
4
6
8
10
PS D:\Full stack\Express js>
```

Justification:

The program asks the user to enter a number N and then checks all numbers from 1 to N. For each number, it tests whether it is even by checking if it is divisible by 2. If the number is even, it is printed. In this way, the program displays all even numbers between 1 and N.

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.”

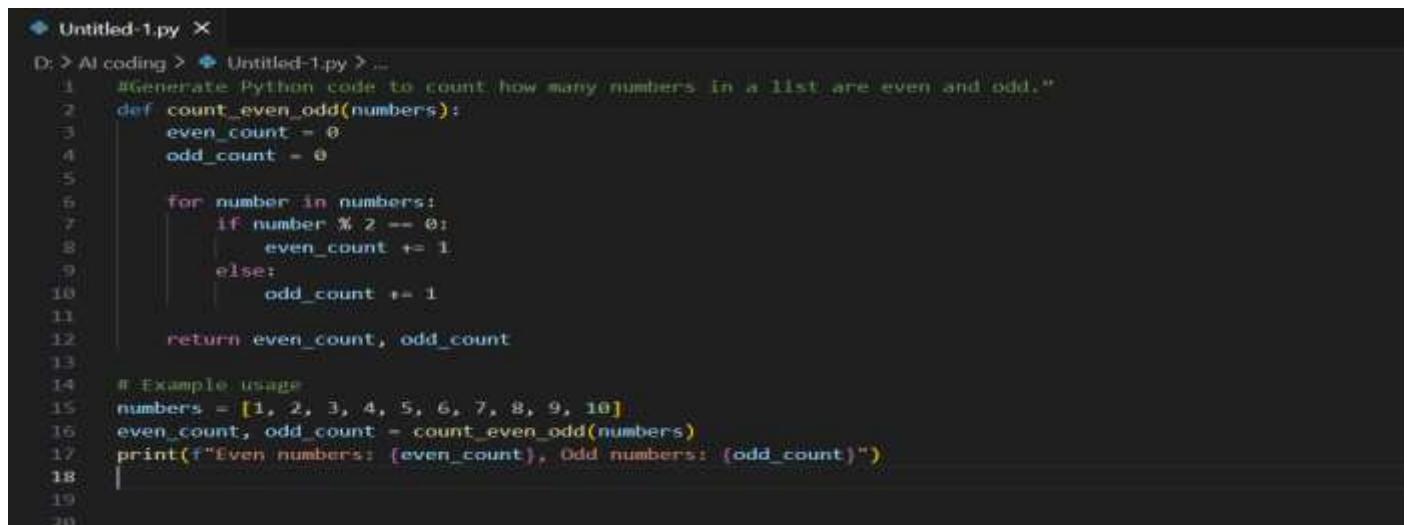
Expected Output:

- AI-generated code using loop and if condition.
- Correct count validation.
- Explanation of logic flow.

Prompt:

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

Code:



```
Untitled-1.py X
D:> AI coding > Untitled-1.py >...
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
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
14 #Example usage
15 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
16 even_count, odd_count = count_even_odd(numbers)
17 print(f"Even numbers: {even_count}, Odd numbers: {odd_count}")
18
19
20
```

Output:

```
PS D:\Full stack\Express js> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/Untitled-1.py"
Even numbers: 5, Odd numbers: 5
PS D:\Full stack\Express js> []
X ⑧ 0 △ 0
```

Justification

The function count_even_odd is used to count how many even and odd numbers are present in a given list. It starts by setting two variables, even_count and odd_count, to zero. The function then goes through each number in the list one by one. If a number is divisible by 2, it is counted as even and even_count is increased. Otherwise, it is counted as odd and odd_count is increased. Finally, the function returns both counts, and the program prints the total number of even and odd values in the list.

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.”

Expected Output:

- AI-generated class with validation logic.
- Verification of condition handling.
- Test cases for valid and invalid inputs

Prompt:

#Generate a Python class User that validates age and email using conditional statements.”

Code

```
Untitled-1.py X
D:\AI coding\Untitled-1.py led-1.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
8      def validate_age(self):
9          if self.age >= 18:
10              print("Age is valid")
11          else:
12              print("Age is not valid (must be 18 or above)")
13
14      def validate_email(self):
15          if "@" in self.email and "." in self.email:
16              print("Email is valid")
17          else:
18              print("Email is not valid")
19
20  # Taking input from user
21  name = input("Enter name: ")
22  age = int(input("Enter age: "))
23  email = input("Enter email: ")
24
25  user = User(name, age, email)
26  user.validate_age()
27  user.validate_email()
28 |
```

Output:

```
PS D:\Full stack\Express js> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/Untitled-1.py"
Enter name: Vineesha
Enter age: 20
Enter email: vineesha@gmail.com
Age is valid
Email is valid
PS D:\Full stack\Express js>
```

Justification:

The program defines a User class that takes name, age, and email as input from the user. It uses conditional statements to check whether the age is 18 or above and whether the email contains both "@" and ".". If the conditions are satisfied, the program displays that the age and email are valid; otherwise, it displays appropriate error messages. This ensures basic validation of user details using simple logic.

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.”

Expected Output:

- AI-generated class code.
- Verification of correctness and completeness of class structure.
- Minor manual improvements (if needed) with justification.

Prompt:

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

Code:

```
Untitled-1.py X
D:\AI coding\Untitled-1.py (ed-1.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 integers
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     # Example usage:
17     student1 = Student("Alice", 101, [85, 90, 78, 92])
18     total_marks = student1.calculate_total()
19     average_marks = student1.calculate_average()
20     print("Total Marks: ", total_marks)
21     print("Average Marks: ", average_marks)
```

Output:

```
Average Marks: 86.25
PS D:\Full stack\Express js> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/Untitled-1.py"
"
Total Marks: 345
Average Marks: 86.25
PS D:\Full stack\Express js> []
```

Justification:

The given program defines a `Student` class to store and process student details such as name, roll number, and marks. The constructor initializes these values, where marks are stored as a list of integers. The `calculate_total` method calculates the total marks by adding all the values in the marks list. The `calculate_average` method finds the average marks by dividing the total marks by the number of subjects, and it also handles the case where no marks are present by returning zero. Finally, an object of the class is created, and the total and average marks are calculated and displayed, showing how the class works.

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.”

Expected Output:

- Complete AI-generated program.
- Identification of strengths and limitations of AI suggestions.
- Reflection on how AI assisted coding productivity.

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

Code:

The screenshot shows a Python code editor interface with a dark theme. The main area displays a script named 'Untitled-1.py' containing code for a BankAccount class and a main() function. The code includes methods for deposit, withdraw, and get_balance, along with a menu loop. The 'EXPLORE' and 'OUTLINE' tabs are visible on the right.

```
D:\AI coding\Untitled-1.py [ed-1.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, balance=0):
4          self.account_holder = account_holder
5          self.balance = balance
6
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
14     def withdraw(self, amount):
15         if amount > 0:
16             if amount <= self.balance:
17                 self.balance -= amount
18                 print(f"Withdraw: ${amount:.2f}")
19             else:
20                 print("Insufficient funds.")
21         else:
22             print("Withdrawal amount must be positive.")
23
24     def get_balance(self):
25         return self.balance
26
27 def main():
28     print("Welcome to the Simple Bank Account System")
29     name = input("Enter account holder's name: ")
30     account = BankAccount(name)
31
32     while True:
33         print("\nMenu:")
34         print("1. Deposit")
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

The screenshot shows a Python code editor interface with a dark theme. The main area displays a completed version of the script 'Untitled-1.py'. It features a main() function that initializes a BankAccount object and then enters a menu loop with options for deposit, withdraw, check balance, and exit. The code uses float() for input amounts and includes a check for the __name__ variable at the bottom. The 'EXPLORE' and 'OUTLINE' tabs are visible on the right.

```
D:\AI coding\Untitled-1.py [ed-1.py] ...
1  def main():
2      name = input("Enter account holder's name: ")
3      account = BankAccount(name)
4
5      while True:
6          print("\nMenu:")
7          print("1. Deposit")
8          print("2. Withdraw")
9          print("3. Check Balance")
10         print("4. Exit")
11
12         choice = input("Choose an option (1-4): ")
13
14         if choice == '1':
15             amount = float(input("Enter amount to deposit: "))
16             account.deposit(amount)
17         elif choice == '2':
18             amount = float(input("Enter amount to withdraw: "))
19             account.withdraw(amount)
20         elif choice == '3':
21             balance = account.get_balance()
22             print(f"Current balance: ${balance:.2f}")
23         elif choice == '4':
24             print("Thank you for using the Simple Bank Account System. Goodbye!")
25             break
26         else:
27             print("Invalid choice. Please select a valid option.")
28
29 if __name__ == "__main__":
30     main()
```

Output:

```
Welcome to the Simple Bank Account System  
Enter account holder's name: vineesha
```

```
Menu:
```

- 1. Deposit
- 2. Withdraw
- 3. Check Balance
- 4. Exit

```
Choose an option (1-4): 2
```

```
Enter amount to withdraw: 2000
```

```
Insufficient funds.
```

```
Menu:
```

- 1. Deposit
- 2. Withdraw
- 3. Check Balance
- 4. Exit

```
Choose an option (1-4): 3
```

```
Current balance: $0.00
```

```
Menu:
```

- 1. Deposit
- 2. Withdraw
- 3. Check Balance
- 4. Exit

```
Choose an option (1-4): 1
```

```
Enter amount to deposit: 10
```

```
Deposited: $10.00
```

Justification:

The given program implements a simple bank account system using a `BankAccount` class. The class stores the account holder's name and current balance, and provides methods to deposit money, withdraw money, and check the balance. Conditional statements are used to ensure that deposit and withdrawal amounts are positive and that withdrawals do not exceed the available balance. The main function takes input from the user, displays a menu with different banking options, and allows the user to perform transactions repeatedly using a loop until they choose to exit. This program demonstrates the use of classes, methods, conditionals, and user interaction to simulate basic banking operations.