Assignment 3.1

AI Assisted Coding

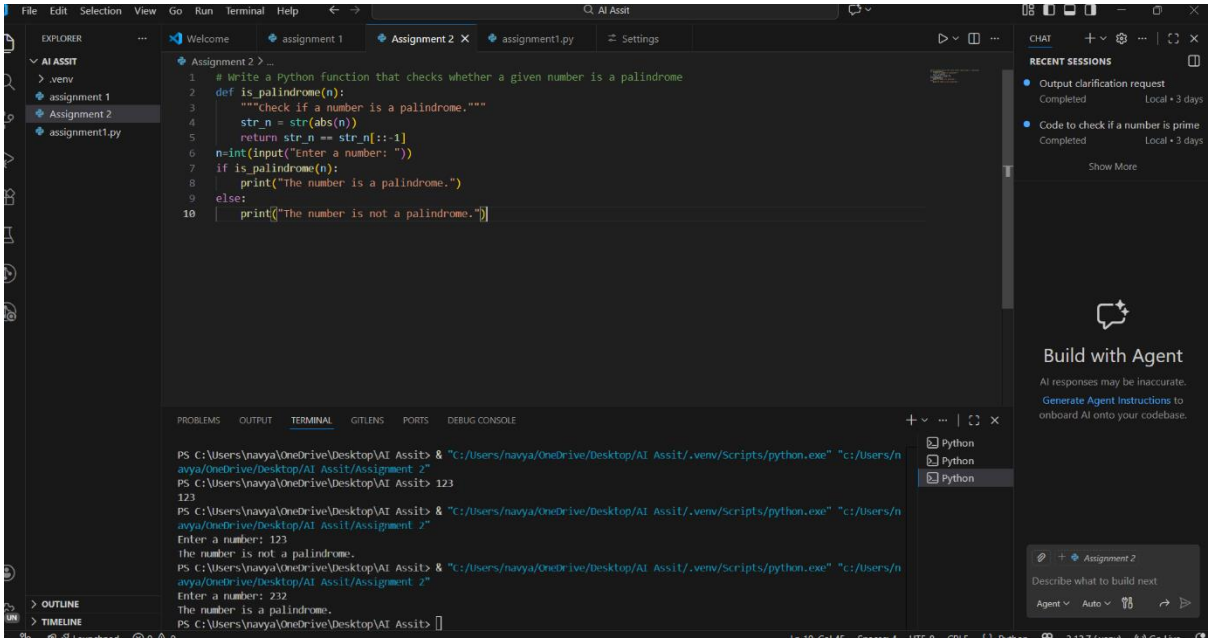
Name: Sunny Kolluri

HTNO: 2303A52444

Task 1:

Prompt:

Write a Python function that checks whether a given number is a palindrome



Observation:

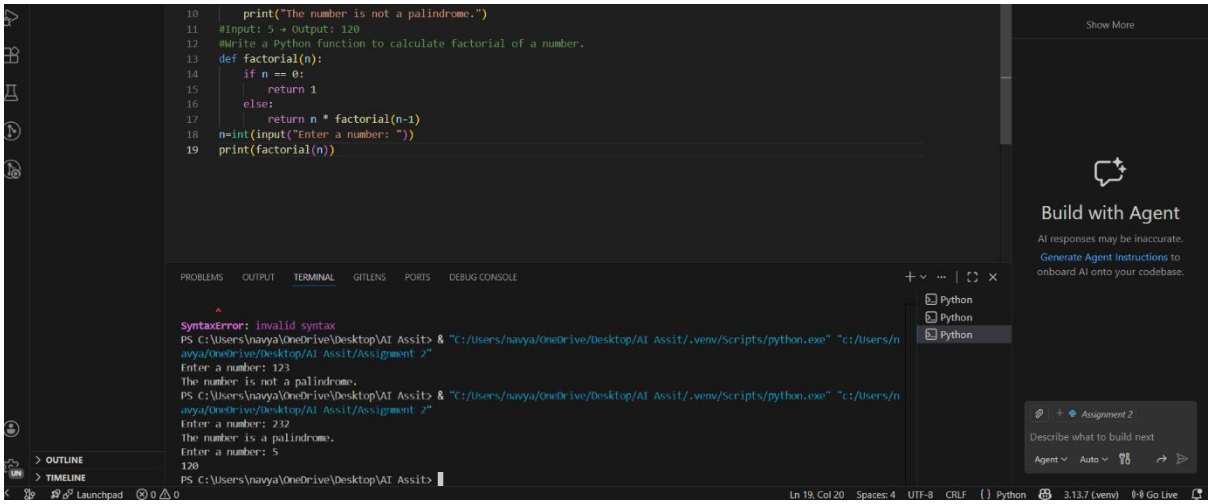
- The program checks whether a given number is a palindrome by converting it into a string.
- The abs(n) function ensures that negative numbers are handled correctly.
- The string is compared with its reverse (str_n[::-1]).
- If both are the same, the number is identified as a palindrome; otherwise, it is not a palindrome.

- This approach is easy to implement, efficient, and avoids complex mathematical operations.

Task 2:

Prompt: #Input: 5 → Output: 120

#Write a Python function to calculate factorial of a number.



Observation:

The program calculates the **factorial of a number** using **recursion**.

The base case n == 0 returns 1, which stops the recursive calls. For values greater than 0, the function calls itself with n-1.

The result is obtained by multiplying all numbers from n down to 1.

This method clearly demonstrates the **concept of recursion** in Python.

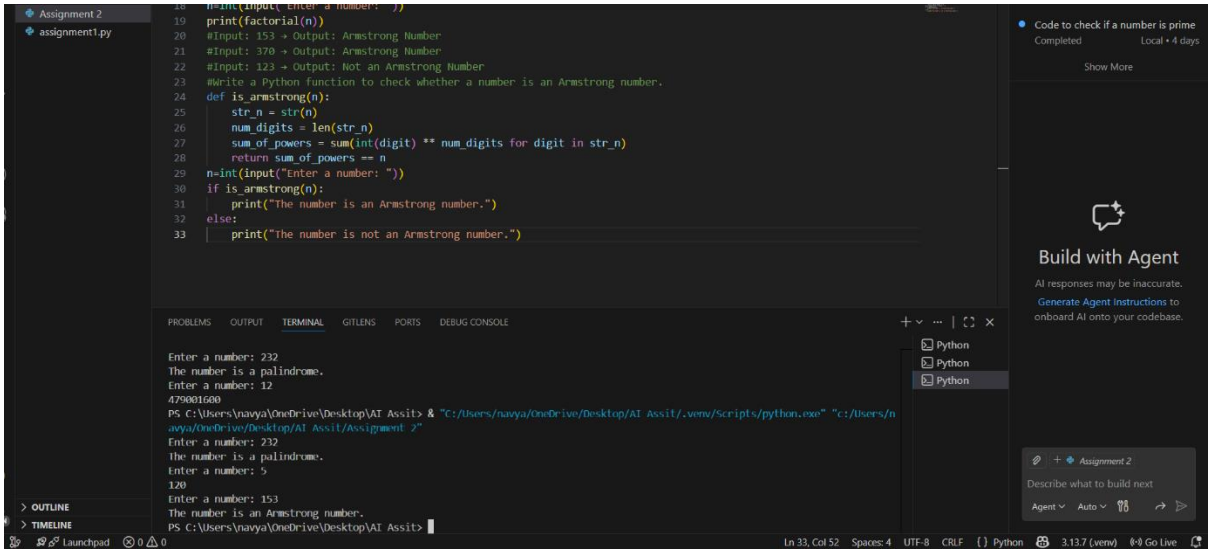
Task3:

Prompt: #Input: 153 → Output: Armstrong Number

#Input: 370 → Output: Armstrong Number

#Input: 123 → Output: Not an Armstrong Number

#Write a Python function to check whether a number is an Armstrong number.



Observation:

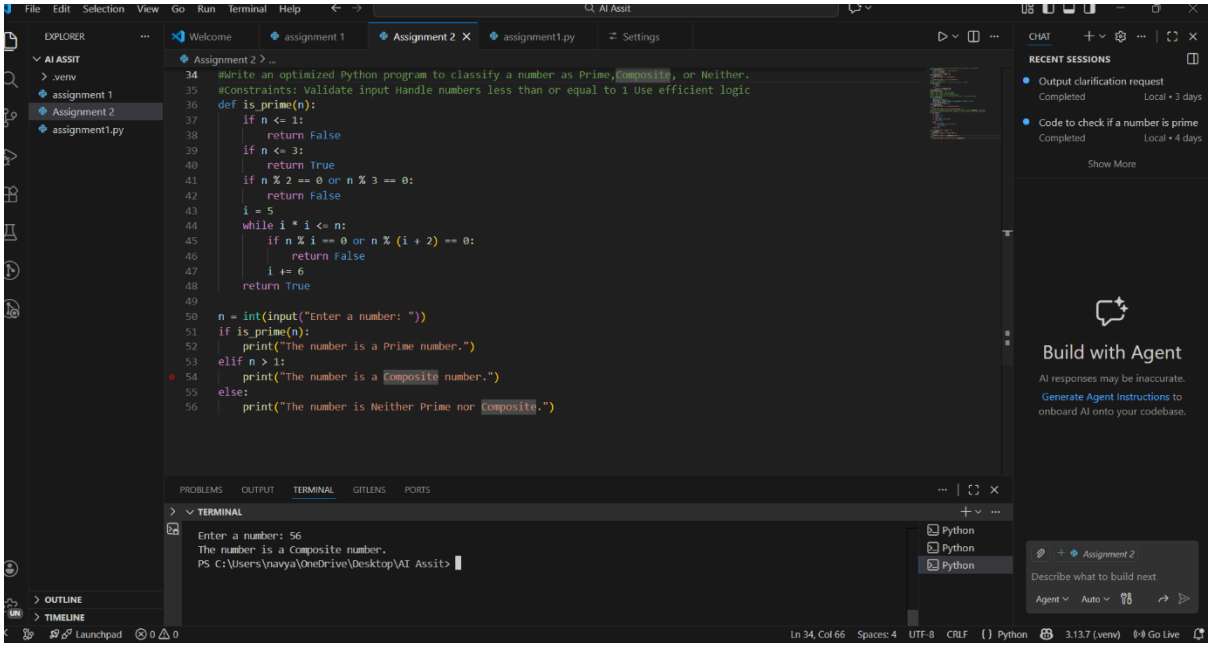
- The program checks whether a given number is an Armstrong number.
- The number is first converted into a string to count the total number of digits.
- Each digit is raised to the power of the total number of digits and added together.
- If the sum of these powers is equal to the original number, it is identified as an Armstrong number.
- This approach is straightforward and avoids complex calculations**.**

Task 4:

Prompt:

#Write an optimized Python program to classify a number as Prime, Composite, or Neither.

#Constraints: Validate input Handle numbers less than or equal to 1 Use efficient logic



Observation:

The program classifies a given number as Prime, Composite, or Neither. Numbers less than or equal to 1 are correctly identified as Neither prime nor composite.

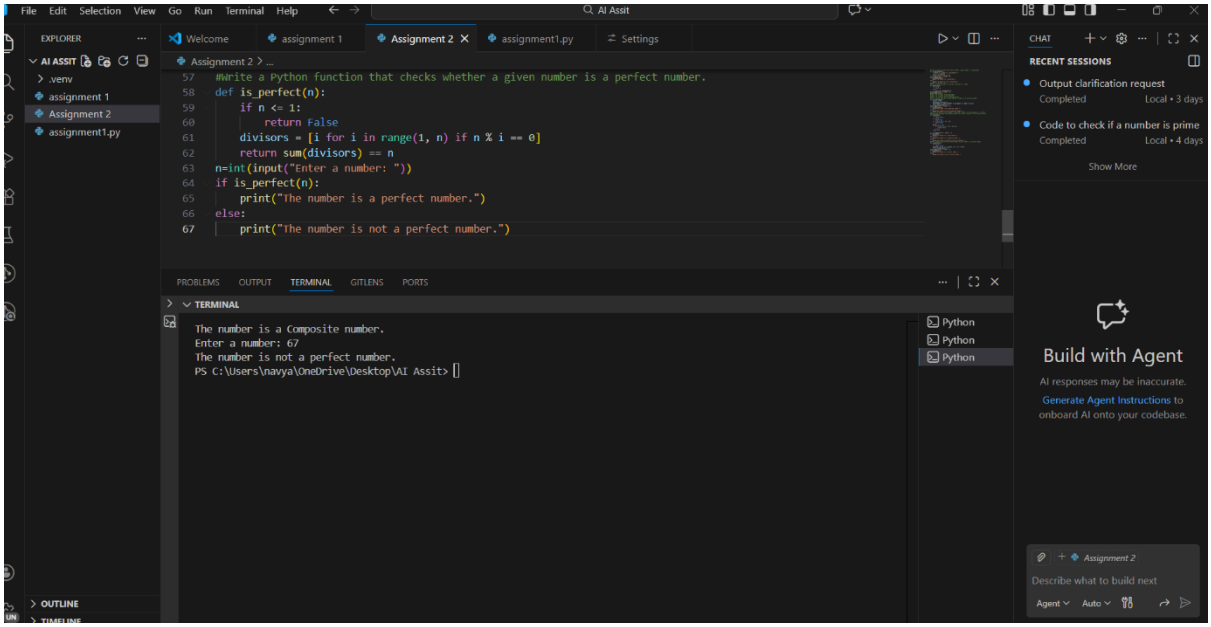
The function uses an optimized prime-checking logic by testing divisibility only up to √n. It skips unnecessary checks by eliminating multiples of 2 and 3 and then checking numbers of the form 6k ± 1.

This approach improves efficiency and reduces execution time for large numbers.

Task 5:

Prompt:

#Write a Python function that checks whether a given number is a perfect number.



Observation:

- The program checks whether a given number is a **perfect number**.
- Numbers less than or equal to **1** are immediately excluded.
- All **proper divisors** of the number (excluding the number itself) are collected.
- The sum of these divisors is compared with the original number.
- If both are equal, the number is identified as a **perfect number**.

Task 6:

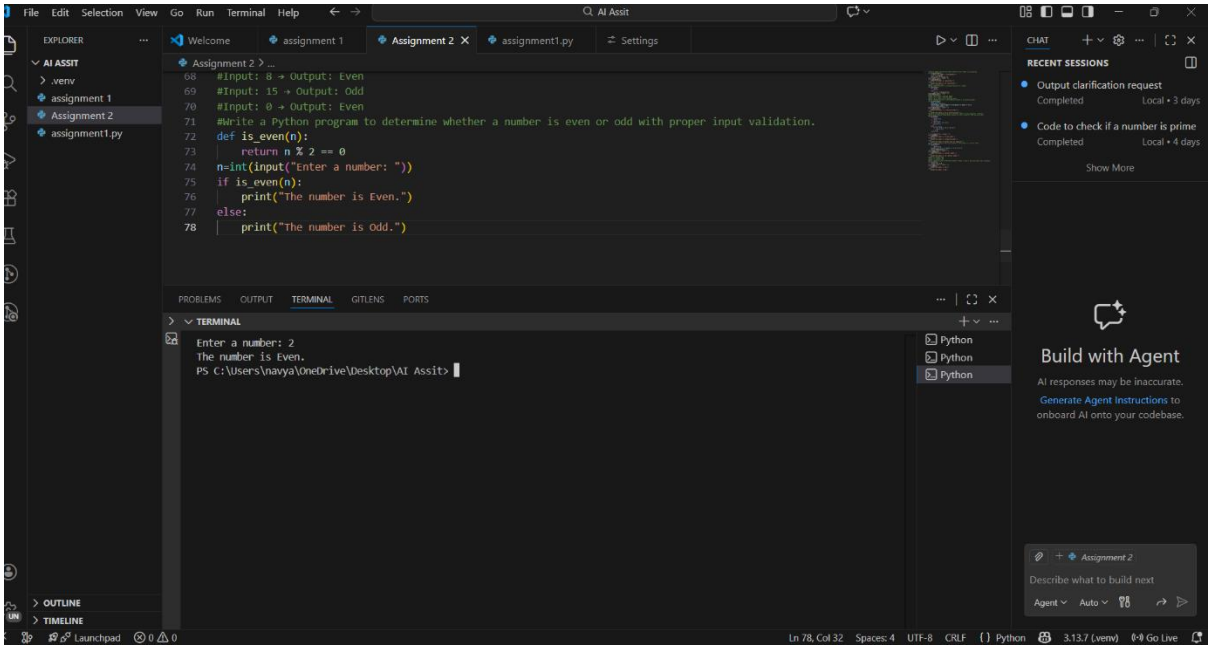
Prompt:

#Input: 8 → Output: Even

#Input: 15 → Output: Odd

#Input: 0 → Output: Even

#Write a Python program to determine whether a number is even or odd with proper input validation.



Observation:

- The program determines whether a given number is even or odd using the modulo (%) operator.
- If a number gives a remainder of 0 when divided by 2, it is classified as Even.
- Otherwise, the number is classified as Odd.
- The program correctly identifies 0 as an even number.
- This method is efficient and works for all integer inputs.