



School of Computer Science and Artificial Intelligence

Lab Assignment-03

Course Title : **AI Assistant Coding**

Name of Student : **Shaik Naved Ahmed**

Enrollment No. : **2303A54053**

Batch No. : **47-B**

QUESTION 1: Zero-Shot Prompting (Palindrome Number Program)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

PROMPT:

Write a Python function that checks whether a given integer is a palindrome. The function should return True if the number is a palindrome and False otherwise. Do not convert the number to a string.

SCREENSHOT OF GENERATED CODE:

The screenshot shows a code editor interface with a dark theme. The left sidebar shows a file tree with 'EXPLORER' and 'AIAC' folder containing 'lab-3.1.py' and 'temp.pages'. The main editor window shows the following Python code:

```
1  #Write a Python function that checks whether a given integer is a palindrome.The function should return True
2  def is_palindrome(num):
3      if num < 0:
4          return False
5      original_num = num
6      reversed_num = 0
7      while num > 0:
8          digit = num % 10
9          reversed_num = reversed_num * 10 + digit
10         num //= 10
11     return original_num == reversed_num
12 # Example usage:
13 print(is_palindrome(121)) # True
14 print(is_palindrome(-121)) # False
15 print(is_palindrome(10)) # False
16 print(is_palindrome(12321))# True
```

The bottom terminal window shows the output of running the script:

```
/usr/local/bin/python3 /Users/navedahmedshaik/Desktop/AIAC/lab-3.1.py
● (base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Desktop/AIAC/lab-3.1.py
True
False
False
True
○ (base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

On the right, there is a 'CHAT' sidebar with a 'Build with Agent' section. It says 'Build with Agent' and 'AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.' Below this is a text input field with placeholder text 'Describe what to build next' and an 'Agent' dropdown set to 'Auto'.

INPUT & OUTPUT:

Input	Output	Reason
121	TRUE	Reversed number is 121, which is equal to the original number.
-121	FALSE	Negative numbers are not considered palindromes.
10	FALSE	Reversed number is 01 (1), which is not equal to 10.
12321	TRUE	Reversed number is 12321, which matches the original number.

EXPLANATION:

The function first checks if the number is negative and immediately returns False because negative numbers cannot be palindromes. It then stores the original number in a separate variable for comparison later and initializes a variable to build the reversed number. Using a while loop, it repeatedly takes the last digit of the number with the modulo operator, appends this digit to the reversed number by multiplying the current reversed value by 10 and adding the digit, and then removes the last digit from the original number using integer division. This process continues until all digits are processed. Finally, the function compares the reversed number with the original number and returns True if they are the same, indicating a palindrome, or False otherwise.

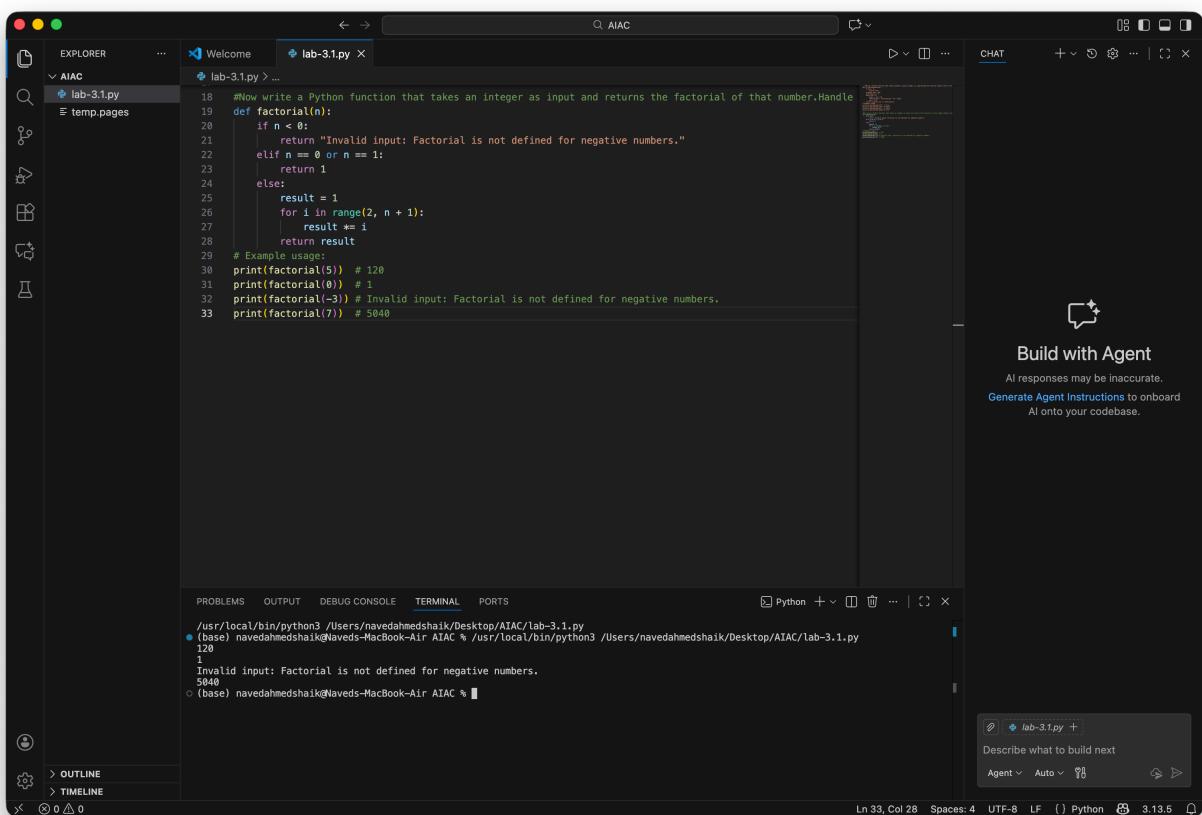
Question 2: One-Shot Prompting (Factorial Calculation)

Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

PROMPT:

Now write a Python function that takes an integer as input and returns the factorial of that number. Handle invalid inputs such as negative numbers appropriately.

SCREENSHOT OF GENERATED CODE:



The screenshot shows a code editor interface with a dark theme. The left sidebar shows a file tree with 'EXPLORER' and 'AIAC' sections, and a file named 'lab-3.1.py' is open. The code editor window contains the following Python code:

```
18 #Now write a Python function that takes an integer as input and returns the factorial of that number.Handle
19 def factorial(n):
20     if n < 0:
21         return "Invalid input: Factorial is not defined for negative numbers."
22     elif n == 0 or n == 1:
23         return 1
24     else:
25         result = 1
26         for i in range(2, n + 1):
27             result *= i
28         return result
29 # Example usage:
30 print(factorial(5)) # 120
31 print(factorial(0)) # 1
32 print(factorial(-3)) # Invalid input: Factorial is not defined for negative numbers.
33 print(factorial(7)) # 5040
```

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

```
120
1
Invalid input: Factorial is not defined for negative numbers.
5040
```

The terminal window has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active. The status bar at the bottom shows 'Ln 33, Col 28' and 'Spaces: 4'.

On the right side of the interface, there is a 'CHAT' window with a message bubble icon and the text 'Build with Agent'. Below it, a note says 'AI responses may be inaccurate.' and 'Generate Agent Instructions to onboard AI onto your codebase.'

INPUT & OUTPUT:

Input	Output	Reason
5	120	$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.
0	1	By definition, $0!$ is equal to 1.
-3	Invalid input: Factorial is not defined for negative numbers.	Factorial is not defined for negative numbers.
7	5040	$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$.

EXPLANATION:

The function first checks if the input number is negative and returns an error message because factorial is not defined for negative numbers. If the number is 0 or 1, it directly returns 1 since the factorial of both is 1. For any positive number greater than 1, it initializes a result variable to 1 and multiplies it by each integer from 2 up to the given number using a loop. After completing the loop, it returns the final computed factorial value.

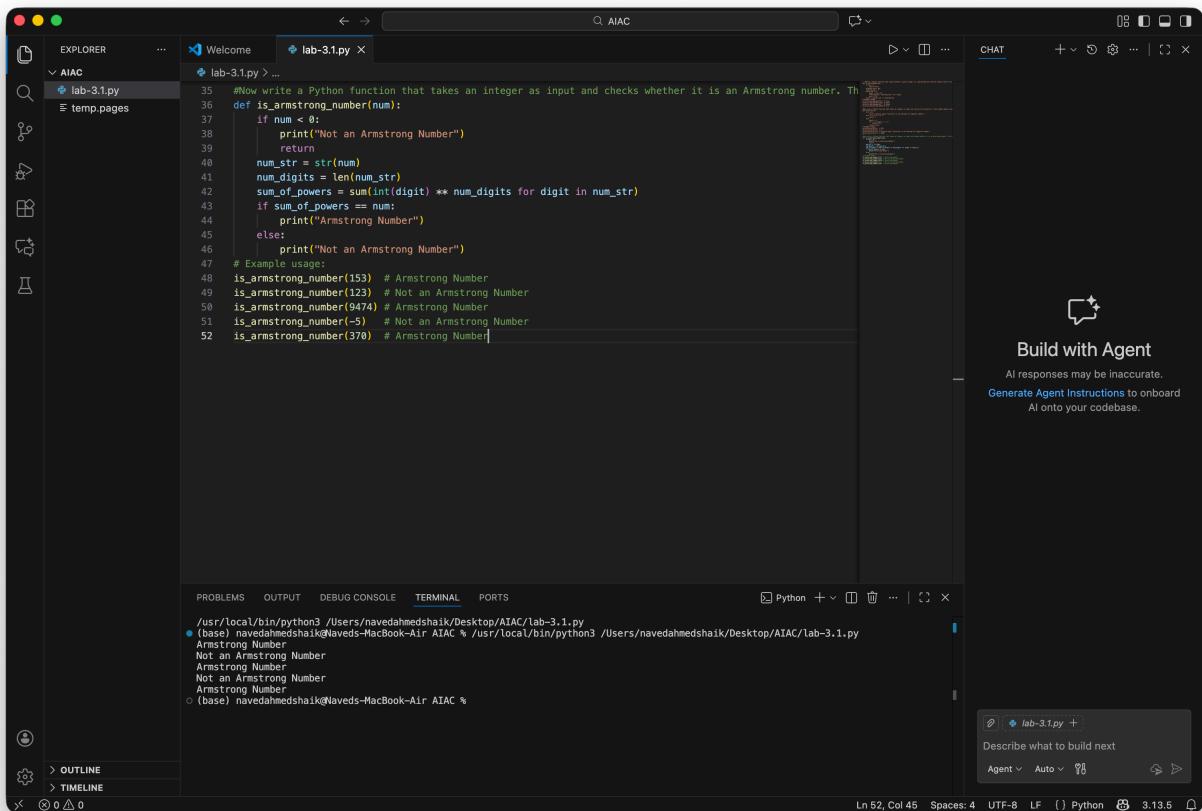
Question 3: Few-Shot Prompting (Armstrong Number Check)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

PROMPT:

Now write a Python function that takes an integer as input and checks whether it is an Armstrong number. The function should print "Armstrong Number" if it is, otherwise "Not an Armstrong Number".

SCREENSHOT OF GENERATED CODE:



The screenshot shows a code editor interface with a dark theme. The left sidebar shows a file tree with 'AIAC' and 'temp.pages' under 'EXPLORER'. The main editor area contains a Python file named 'lab-3.1.py' with the following code:

```
35 #Now write a Python function that takes an integer as input and checks whether it is an Armstrong number. Then
36 def is_armstrong_number(num):
37     if num < 0:
38         print("Not an Armstrong Number")
39         return
40     num_str = str(num)
41     num_digits = len(num_str)
42     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
43     if sum_of_powers == num:
44         print("Armstrong Number")
45     else:
46         print("Not an Armstrong Number")
47 # Example usage:
48 is_armstrong_number(153) # Armstrong Number
49 is_armstrong_number(123) # Not an Armstrong Number
50 is_armstrong_number(9474) # Armstrong Number
51 is_armstrong_number(-5) # Not an Armstrong Number
52 is_armstrong_number(370) # Armstrong Number
```

The bottom terminal window shows the execution of the script:

```
/usr/local/bin/python3 /Users/navedahmedshaik/Desktop/AIAC/lab-3.1.py
• (base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Desktop/AIAC/lab-3.1.py
Armstrong Number
Not an Armstrong Number
Armstrong Number
Not an Armstrong Number
Armstrong Number
• (base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

A sidebar on the right is titled 'Build with Agent' with the sub-instruction 'AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.' Below the terminal, a 'Build' button is visible.

INPUT & OUTPUT:

Inp ut	Output	Reason
153	Armstrong Number	$(1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)$, which equals the original number.
123	Not an Armstrong Number	$(1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36)$, not equal to 123.
9474	Armstrong Number	$(9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474)$.
-5	Not an Armstrong Number	Negative numbers are not considered Armstrong numbers.
370	Armstrong Number	$(3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370)$, which equals the original number.

EXPLANATION:

The function first checks if the number is negative and immediately prints "Not an Armstrong Number" because Armstrong numbers cannot be negative. It then converts the number into a string to count how many digits it has. For each digit, it raises the digit to the power of the total number of digits and adds these values together. If the resulting sum is equal to the original number, the function prints "Armstrong Number"; otherwise, it prints "Not an Armstrong Number".

Question 4: Context-Managed Prompting (Optimized Number Classification)

Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

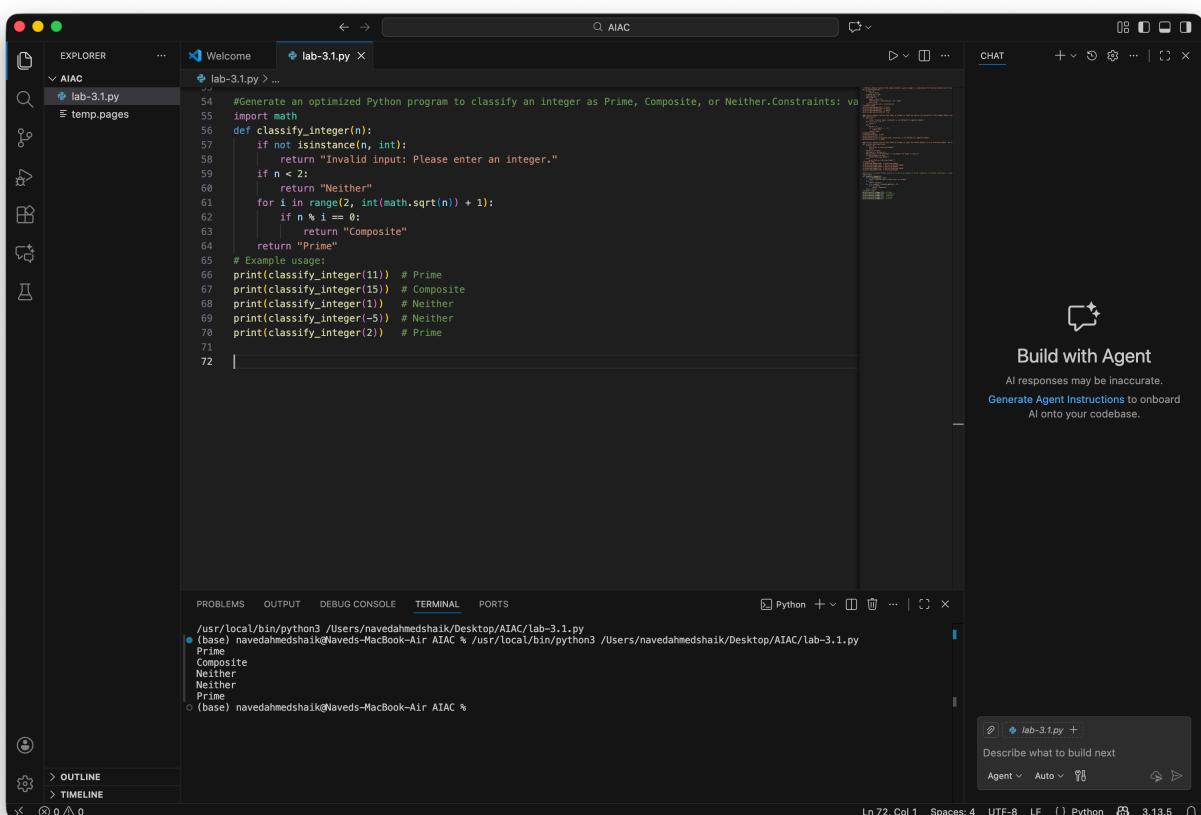
- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

PROMPT:

Generate an optimized Python program to classify an integer as Prime, Composite, or Neither.

Constraints: validate input, check divisibility up to \sqrt{n} , print only one result (Prime, Composite, or Neither), and briefly compare with a basic approach.

SCREENSHOT OF GENERATED CODE:



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

- EXPLORER:** Shows a folder structure with 'AIAC' and files 'lab-3.1.py' and 'temp.pages'.
- CODE EDITOR:** The file 'lab-3.1.py' contains the following Python code:54 #Generate an optimized Python program to classify an integer as Prime, Composite, or Neither. Constraints: validate input, check divisibility up to sqrt(n), print only one result (Prime, Composite, or Neither), and briefly compare with a basic approach.
55 import math
56 def classify_integer(n):
57 if not isinstance(n, int):
58 return "Invalid input: Please enter an integer."
59 if n < 2:
60 return "Neither"
61 for i in range(2, int(math.sqrt(n)) + 1):
62 if n % i == 0:
63 return "Composite"
64 return "Prime"
65 # Example usage:
66 print(classify_integer(11)) # Prime
67 print(classify_integer(15)) # Composite
68 print(classify_integer(1)) # Neither
69 print(classify_integer(-5)) # Neither
70 print(classify_integer(2)) # Prime
71
72
- CHAT:** A sidebar with the heading 'Build with Agent' and the sub-instruction 'AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.'
- TERMINAL:** Shows the command line output of running the script:/usr/local/bin/python3 /Users/navedahmedshaik/Desktop/AIAC/lab-3.1.py
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
Prime
Composite
Neither
Neither
Prime
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
- STATUS BAR:** Shows the line 'Ln 72, Col 1' and other standard status bar information.

INPUT & OUTPUT:

Input	Output	Reason
11	Prime	No divisors found from 2 to $\sqrt{11}$, so it is a prime number.
15	Composite	Divisible by 3 ($15 \% 3 = 0$), so it is a composite number.
1	Neither	Numbers less than 2 are neither prime nor composite.
-5	Neither	Negative numbers are neither prime nor composite.
2	Prime	Only divisible by 1 and itself; loop doesn't find any other divisors.

EXPLANATION:

The function first validates whether the input is an integer; if not, it returns an invalid input message. For integers less than 2, it returns "Neither" because numbers like 0, 1, and negatives are neither prime nor composite. For valid integers ≥ 2 , it checks divisibility only up to the square root of the number, which is more efficient than checking all numbers up to n. If any divisor is found, it returns "Composite"; otherwise, it returns "Prime".

Question 5: Zero-Shot Prompting (Perfect Number Check)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

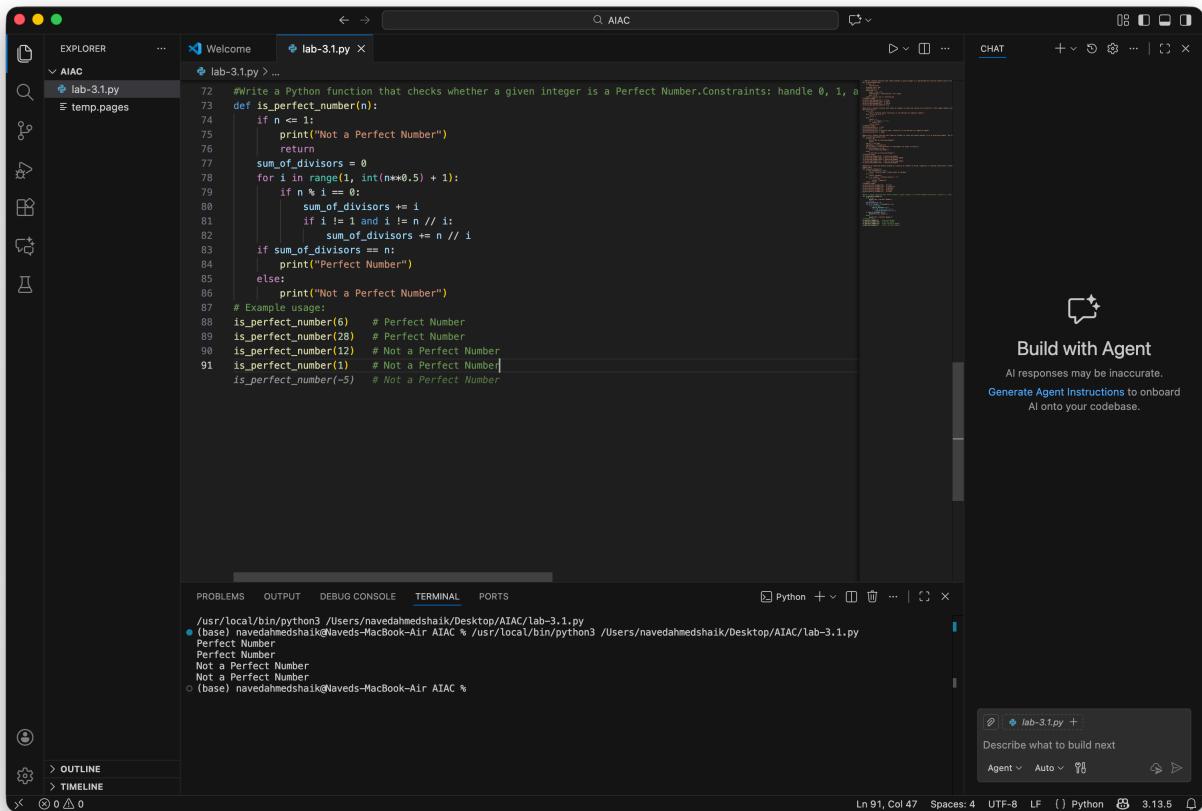
PROMPT:

Write a Python function that checks whether a given integer is a Perfect Number.

Constraints: handle 0, 1, and negative numbers, use an efficient divisor check, and print Perfect Number or Not a Perfect Number.

SCREENSHOT OF GENERATED CODE:

INPUT & OUTPUT:



```
72 #Write a Python function that checks whether a given integer is a Perfect Number. Constraints: handle 0, 1, a
73 def is_perfect_number(n):
74     if n <= 1:
75         print("Not a Perfect Number")
76         return
77     sum_of_divisors = 0
78     for i in range(1, int(n**0.5) + 1):
79         if n % i == 0:
80             sum_of_divisors += i
81             if i != 1 and i != n // i:
82                 sum_of_divisors += n // i
83     if sum_of_divisors == n:
84         print("Perfect Number")
85     else:
86         print("Not a Perfect Number")
87 # Example usage:
88 is_perfect_number(6) # Perfect Number
89 is_perfect_number(28) # Perfect Number
90 is_perfect_number(12) # Not a Perfect Number
91 is_perfect_number(1) # Not a Perfect Number
92 is_perfect_number(-5) # Not a Perfect Number
```

The screenshot shows a code editor with a Python file named 'lab-3.1.py' open. The code defines a function 'is_perfect_number' that checks if a given integer is a perfect number. It handles edge cases for 0 and 1. For other numbers, it calculates the sum of divisors (excluding the number itself) and compares it to the original number. The code uses a for loop to iterate from 1 to the square root of the number, checking for divisors. If a divisor is found, it adds both the divisor and its pair to the sum. Finally, it prints 'Perfect Number' if the sum equals the original number, or 'Not a Perfect Number' otherwise. The terminal below shows the execution of the script with various inputs and the resulting output.

Input	Output	Reason
6	Perfect Number	Divisors: 1, 2, 3 → Sum = 6, which equals the number.
28	Perfect Number	Divisors: 1, 2, 4, 7, 14 → Sum = 28, which equals the number.
12	Not a Perfect Number	Divisors: 1, 2, 3, 4, 6 → Sum = 16, not equal to 12.
1	Not a Perfect Number	1 has no proper divisors that sum to itself.
0	Not a Perfect Number	0 is not considered a perfect number.
-5	Not a Perfect Number	Negative numbers are not considered perfect numbers.

EXPLANATION:

The function first checks if the number is less than or equal to 1 and prints "Not a Perfect Number" because perfect numbers are positive integers greater than 1. It then initializes a variable to store the sum of proper divisors. Using an efficient loop that runs only up to the square root of the number, it finds divisors in pairs. For each divisor found, it adds both the divisor and its pair (if they are different and not the number itself) to the sum. Finally, it compares the sum of divisors with the original number and prints "Perfect Number" if they are equal; otherwise, it prints "Not a Perfect Number".

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

Task:

- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

PROMPT:

Write a Python program to check if a number is Even or Odd with input validation.

Examples:

Input: 8 → Output: Even

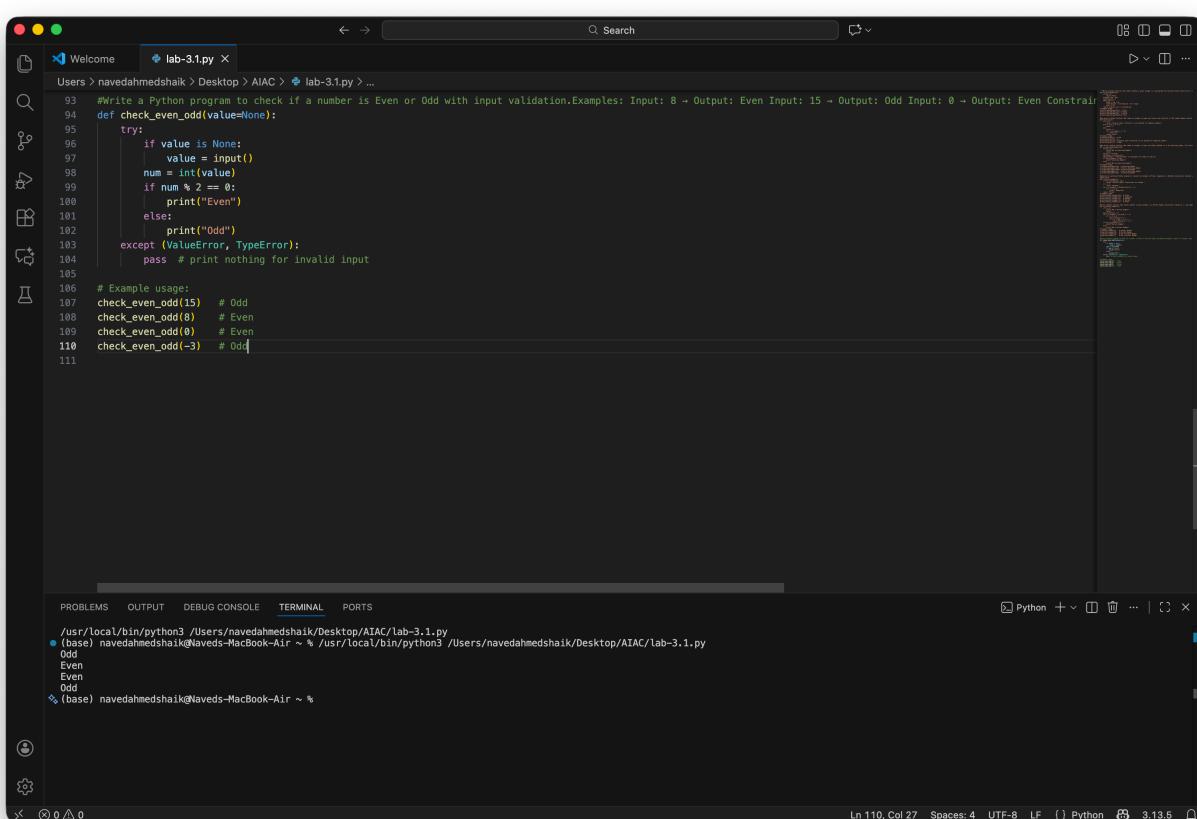
Input: 15 → Output: Odd

Input: 0 → Output: Even

Constraints:

- Handle non-integer inputs and negative numbers.
- Print only Even or Odd.

SCREENSHOT OF GENERATED CODE:



```
93 #Write a Python program to check if a number is Even or Odd with input validation.Examples: Input: 8 -> Output: Even Input: 15 -> Output: Odd Input: 0 -> Output: Even Constraints: Input: 8.5 -> Output: None
94 def check_even_odd(value=None):
95     try:
96         if value is None:
97             value = input()
98         num = int(value)
99         if num % 2 == 0:
100             print("Even")
101         else:
102             print("Odd")
103     except (ValueError, TypeError):
104         pass # print nothing for invalid input
105
106 # Example usage:
107 check_even_odd(15) # Odd
108 check_even_odd(8) # Even
109 check_even_odd(0) # Even
110 check_even_odd(-3) # Odd
```

The screenshot shows a code editor with a Python file named 'lab-3.1.py'. The code defines a function 'check_even_odd' that takes an optional parameter 'value'. It uses a try-except block to handle non-integer inputs. If 'value' is not provided, it prompts the user for input. It then converts the input to an integer and checks if it is divisible by 2. If it is, it prints 'Even'; otherwise, it prints 'Odd'. The code also includes a section for example usage, demonstrating the function with various inputs like 15, 8, 0, and -3. The code editor interface includes tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab shows the command-line output of running the script with these inputs, showing the expected 'Even' or 'Odd' output for each case.

INPUT & OUTPUT:

Input	Output	Reason
15	Odd	15 is not divisible by 2 ($15 \% 2 = 1$).
8	Even	8 is divisible by 2 ($8 \% 2 = 0$).
0	Even	0 is divisible by 2 ($0 \% 2 = 0$).
-3	Odd	-3 is not divisible by 2 ($-3 \% 2 = 1$).
"abc"	—	Invalid input; cannot be converted to an integer.

EXPLANATION:

The function checks whether a given input is even or odd. If no value is passed, it takes input from the user. It then tries to convert the input into an integer. If the conversion is successful, it checks divisibility by 2: if the number is divisible by 2, it prints "Even", otherwise it prints "Odd". If the input is invalid (not a number or wrong type), the function catches the error and prints nothing.