SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Dr. Rishabh Mittal
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju

CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week2 – Monday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches

Assignment Number:3.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab Experiment: Prompt Engineering – Improving Prompts and Context Management**	Week2 - Monday

Lab Objectives

1. To understand and apply different prompt engineering techniques for generating Python programs using AI-assisted tools.
2. To analyze the impact of context and examples on the accuracy and efficiency of AI-generated code.
3. To develop and refine real-world Python applications through iterative prompt improvement.

Lab Outcomes

1. Students will be able to design effective prompts to generate correct and optimized Python code.
2. Students will be able to compare and evaluate AI-generated solutions produced using different prompting strategies.
3. Students will be able to implement and document real-world Python applications using AI-assisted coding tools.

Experiment – Prompt Engineering Techniques

Task Description

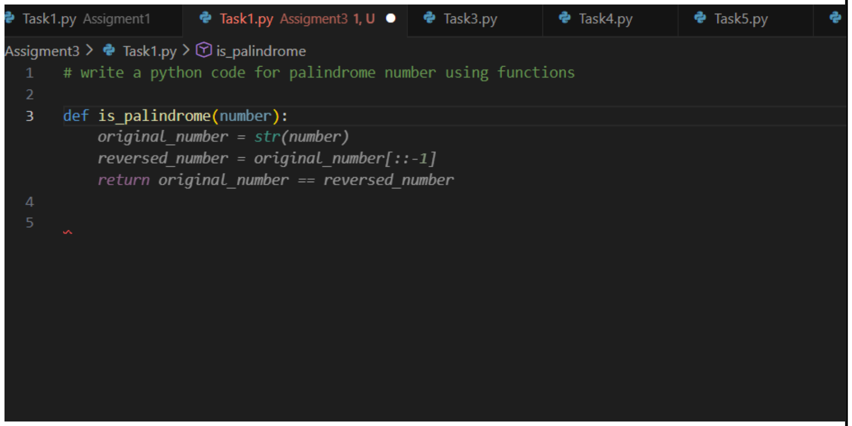
Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.

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.

Task:

- Record the AI-generated code.



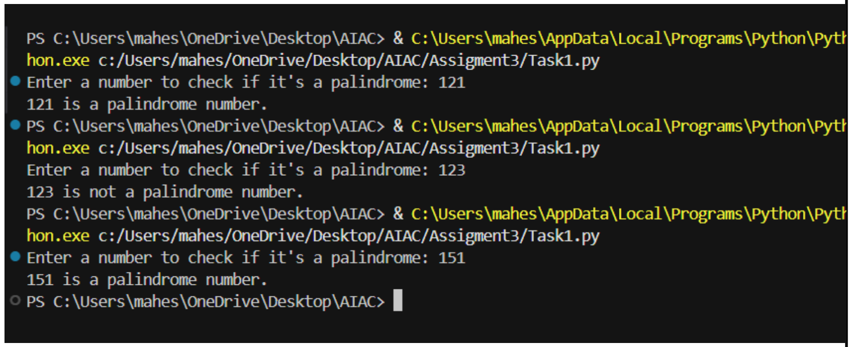
Task1.py Assigment1 | Task1.py Assigment3 1, U | Task3.py | Task4.py | Task5.py

Assigment3 > Task1.py > is_palindrome

```python
# write a python code for palindrome number using functions

def is_palindrome(number):
    original_number = str(number)
    reversed_number = original_number[::-1]
    return original_number == reversed_number
```

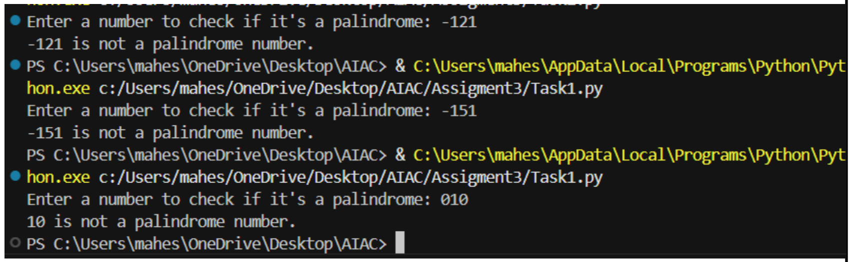
- Test the code with multiple inputs.



```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyth
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task1.py
Enter a number to check if it's a palindrome: 121
121 is a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyth
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task1.py
Enter a number to check if it's a palindrome: 123
123 is not a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyth
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task1.py
Enter a number to check if it's a palindrome: 151
151 is a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

- Identify any logical errors or missing edge-case handling.

  The program does not work correctly when the number starts with 010.



```
Enter a number to check if it's a palindrome: -121
-121 is not a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task1.py
Enter a number to check if it's a palindrome: -151
-151 is not a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task1.py
Enter a number to check if it's a palindrome: 010
10 is not a palindrome number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

Question 2: One-Shot Prompting (Factorial Calculation)

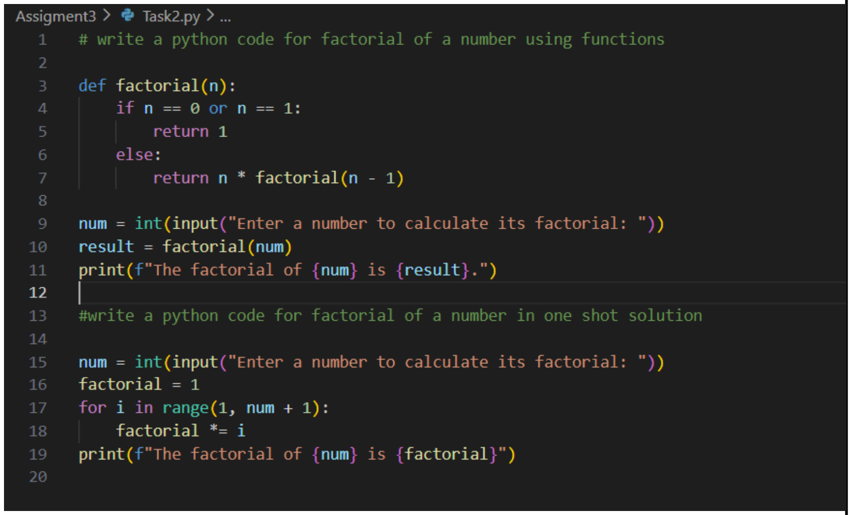
Write a **one-shot prompt** by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 ⟶ Output: 120

Task:

- Compare the generated code with a zero-shot solution.



Assigment3 > Task2.py > ...

```python
# write a python code for factorial of a number using functions

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

num = int(input("Enter a number to calculate its factorial: "))
result = factorial(num)
print(f"The factorial of {num} is {result}.")

#write a python code for factorial of a number in one shot solution

num = int(input("Enter a number to calculate its factorial: "))
factorial = 1
for i in range(1, num + 1):
    factorial *= i
print(f"The factorial of {num} is {factorial}")

```

- Examine improvements in clarity and correctness.

```
The recursive and iterative implementations are
both correct for non-negative inputs, but they
lack input validation, handling for negative
numbers, and explanatory comments, which reduces
robustness and clarity. While the recursive
version risks errors for large inputs due to
recursion limits, the iterative version is more
scalable, though the code overall would benefit
from better structure, documentation, and basic
testing.
```

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.

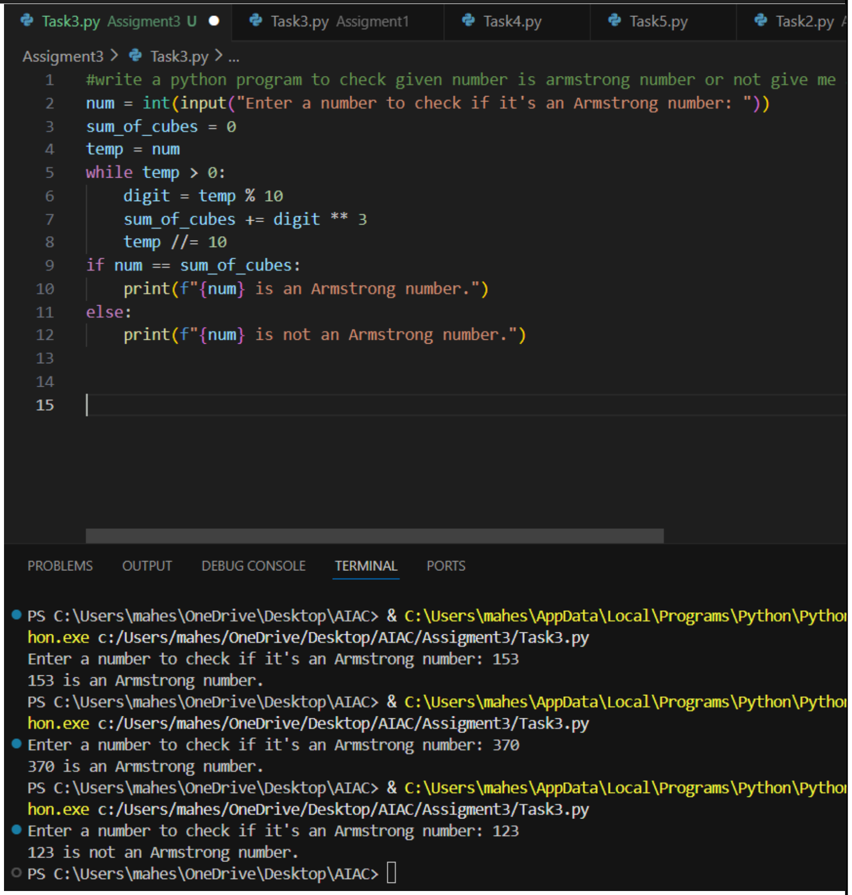
Examples:

- Input: 153 ⟶ Output: Armstrong Number
- Input: 370 ⟶ Output: Armstrong Number
- Input: 123 ⟶ Output: Not an Armstrong Number

Task:

- Analyze how multiple examples influence code structure and accuracy.

```
Uses a while loop to extract digits, compute
cubes, and compare the sum to the original number.
```

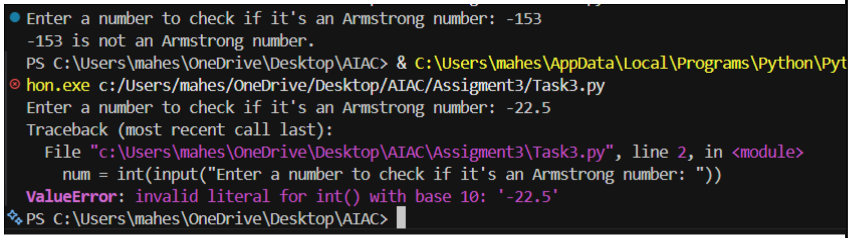


```python
#write a python program to check given number is armstrong number or not give me
num = int(input("Enter a number to check if it's an Armstrong number: "))
sum_of_cubes = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum_of_cubes += digit ** 3
    temp //= 10
if num == sum_of_cubes:
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")


```

```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pytho
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task3.py
Enter a number to check if it's an Armstrong number: 153
153 is an Armstrong number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pytho
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task3.py
Enter a number to check if it's an Armstrong number: 370
370 is an Armstrong number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pytho
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task3.py
Enter a number to check if it's an Armstrong number: 123
123 is not an Armstrong number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

- Test the function with boundary values and invalid inputs.



```
Enter a number to check if it's an Armstrong number: -153
-153 is not an Armstrong number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task3.py
Enter a number to check if it's an Armstrong number: -22.5
Traceback (most recent call last):
  File "c:\Users\mahes\OneDrive\Desktop\AIAC\Assigment3\Task3.py", line 2, in <module>
    num = int(input("Enter a number to check if it's an Armstrong number: "))
ValueError: invalid literal for int() with base 10: '-22.5'
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

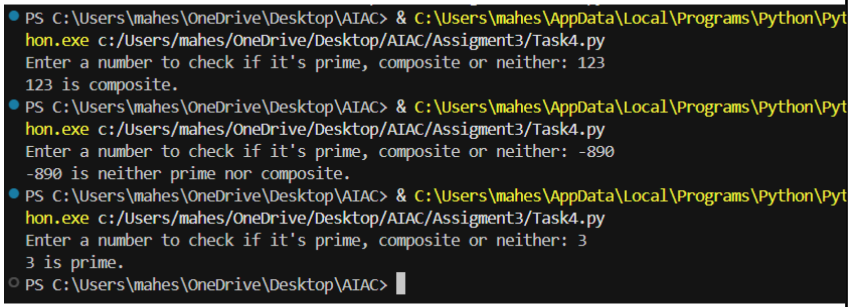
(Optional Extension)

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**.

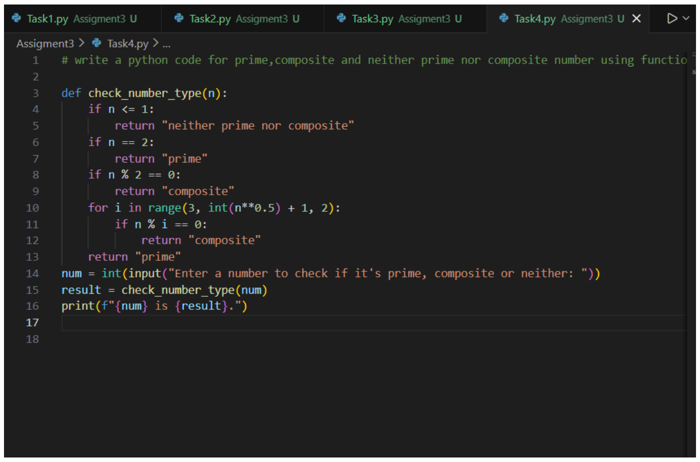
Task:

- Ensure proper input validation.



```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task4.py
Enter a number to check if it's prime, composite or neither: 123
123 is composite.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task4.py
Enter a number to check if it's prime, composite or neither: -890
-890 is neither prime nor composite.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task4.py
Enter a number to check if it's prime, composite or neither: 3
3 is prime.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

- Optimize the logic for efficiency.



```python
# write a python code for prime,composite and neither prime nor composite number using functio

def check_number_type(n):
    if n <= 1:
        return "neither prime nor composite"
    if n == 2:
        return "prime"
    if n % 2 == 0:
        return "composite"
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return "composite"
    return "prime"
num = int(input("Enter a number to check if it's prime, composite or neither: "))
result = check_number_type(num)
print(f"{num} is {result}.")


```

- Compare the output with earlier prompting strategies.

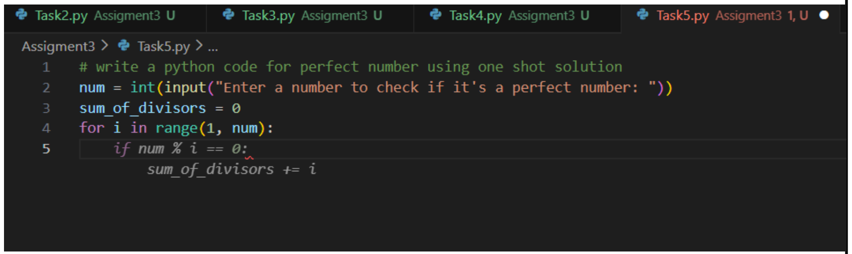
- Input: 2 → "2 is prime."
- Input: 4 → "4 is composite."
- Input: 1 → "1 is neither prime nor composite."
- Input: 29 → "29 is prime." (Efficient, ~3 iterations in loop.)
- Input: abc → "Invalid input. Please enter an integer."
- Vs. Zero-Shot: Might output "2 is prime" but crash on abc. Vs. Few-Shot: Similar accuracy but slower for large n (e.g., 1000003 takes more time without even-check).

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.

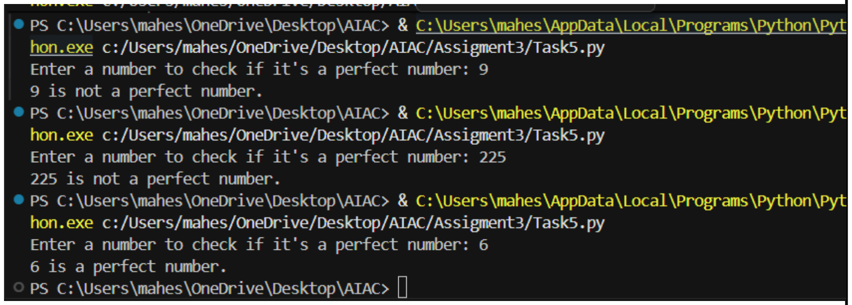
Task:

- Record the AI-generated code.



```python
# write a python code for perfect number using one shot solution
num = int(input("Enter a number to check if it's a perfect number: "))
sum_of_divisors = 0
for i in range(1, num):
    if num % i == 0:
        sum_of_divisors += i
```

- Test the program with multiple inputs.



```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task5.py
Enter a number to check if it's a perfect number: 9
9 is not a perfect number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task5.py
Enter a number to check if it's a perfect number: 225
225 is not a perfect number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task5.py
Enter a number to check if it's a perfect number: 6
6 is a perfect number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> 
```

- Identify any missing conditions or inefficiencies in the logic.

```
No edge-case handling:

For num ≤ 0, the logic is invalid (perfect numbers
are positive integers).

num = 1 should explicitly be treated as not a
perfect number.

Inefficient loop:

The loop checks all numbers from 1 to num-1; it
can be optimized by checking divisors only up to
num // 2 or √num.

No input validation:

Non-integer or invalid input will cause a runtime
error.
```

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.

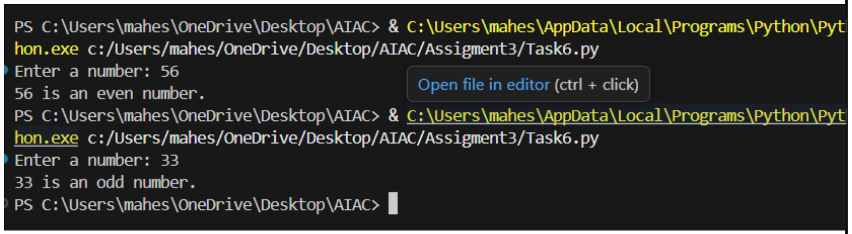
Examples:

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

Task:

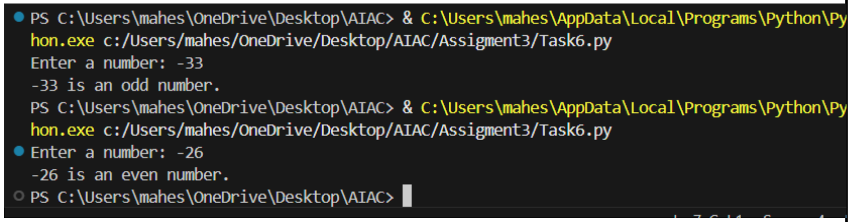
- Analyze how examples improve input handling and output clarity.

```
Examples help explain:
n % 2 == 0 → even number
n % 2 != 0 → odd number
```



```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task6.py
Enter a number: 56
56 is an even number.                                 Open file in editor (ctrl + click)
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyt
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task6.py
Enter a number: 33
33 is an odd number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```

- Test the program with negative numbers and non-integer inputs.



```
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Py
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task6.py
Enter a number: -33
-33 is an odd number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC> & C:\Users\mahes\AppData\Local\Programs\Python\Py
hon.exe c:/Users/mahes/OneDrive/Desktop/AIAC/Assigment3/Task6.py
Enter a number: -26
-26 is an even number.
PS C:\Users\mahes\OneDrive\Desktop\AIAC>
```