AI Assisted coding

Assignment-3.1

V.Snehitha

2303A52331

Batch-39

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

Write a zero-shot prompt (without providing any examples) to generate

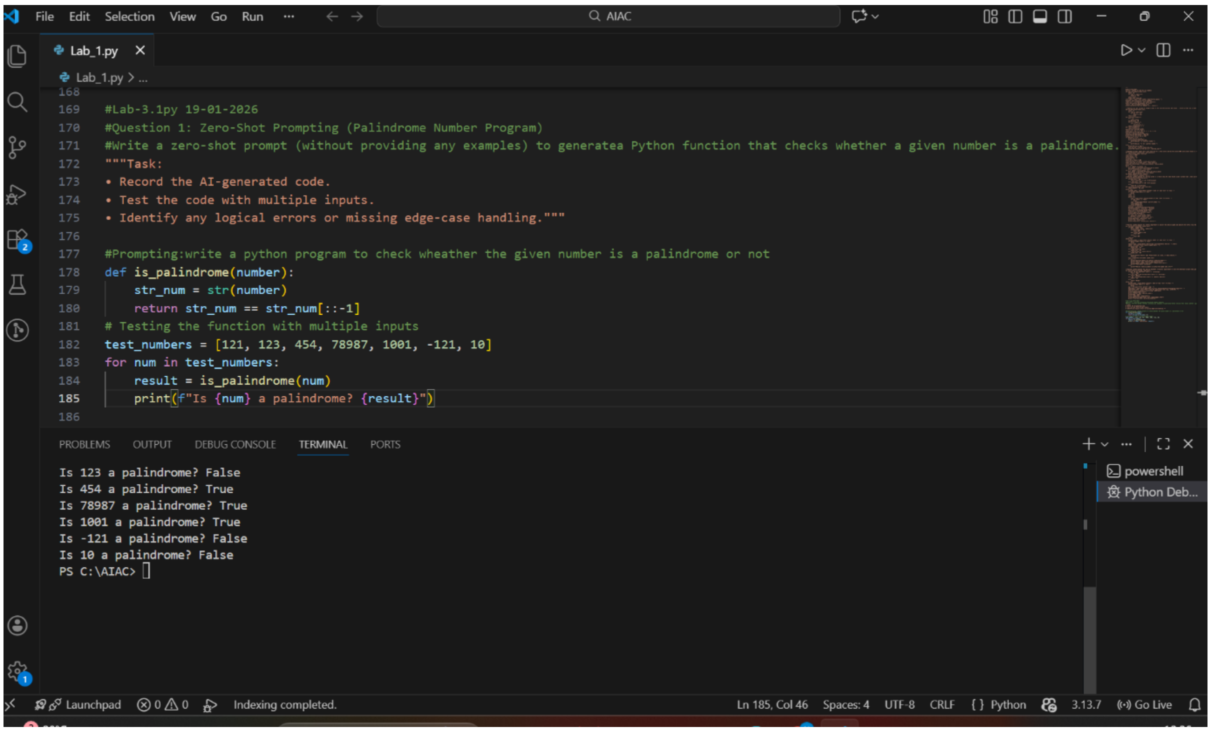
a Python function that checks whether a given number is a palindrome.

Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

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



```python
#Lab-3.1py 19-01-2026
#Question 1: Zero-Shot Prompting (Palindrome Number Program)
#Write a zero-shot prompt (without providing any examples) to generatea Python function that checks whether a given number is a palindrome.
"""Task:
• Record the AI-generated code.
• Test the code with multiple inputs.
• Identify any logical errors or missing edge-case handling."""

#Prompting:write a python program to check wheather the given number is a palindrome or not
def is_palindrome(number):
    str_num = str(number)
    return str_num == str_num[::-1]
# Testing the function with multiple inputs
test_numbers = [121, 123, 454, 78987, 1001, -121, 10]
for num in test_numbers:
    result = is_palindrome(num)
    print(f"Is {num} a palindrome? {result}")
```

```
Is 123 a palindrome? False
Is 454 a palindrome? True
Is 78987 a palindrome? True
Is 1001 a palindrome? True
Is -121 a palindrome? False
Is 10 a palindrome? False
PS C:\AIAC> 
```

Analysis:

Works correctly for basic positive numbers

Negative numbers fail due to string behavior, not real logic

No input type checking is done

Relies only on string conversion

Suitable only for simple or beginner-level tasks

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:

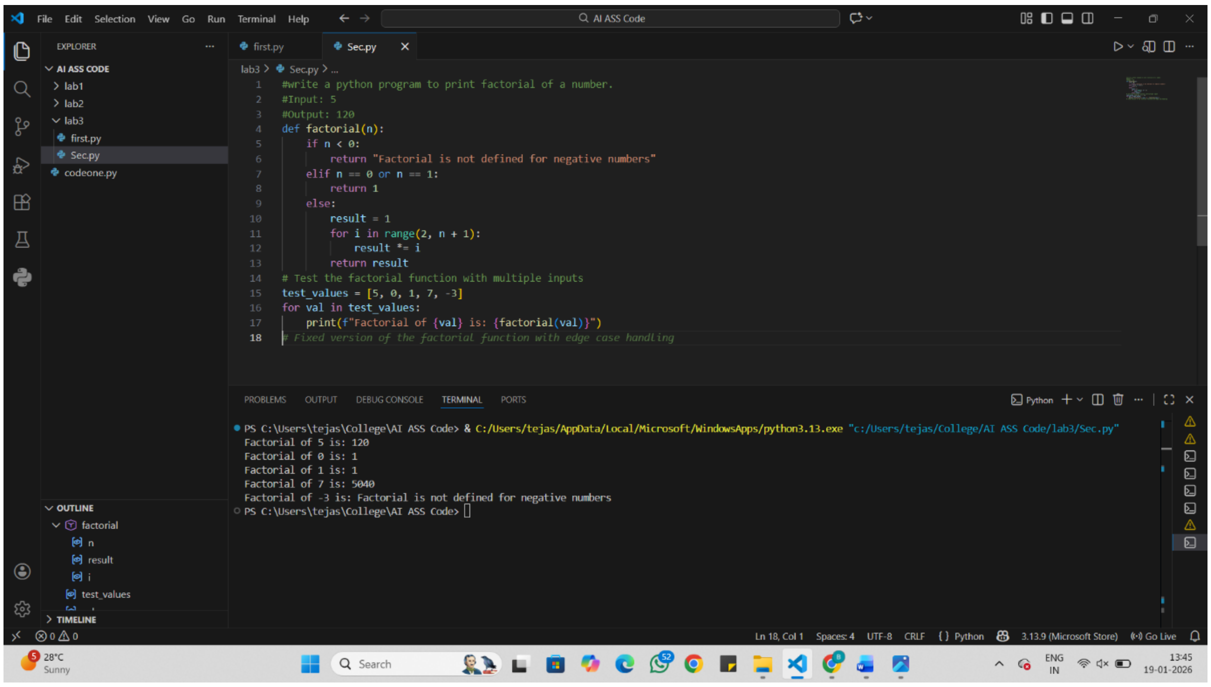
Input: 5 → Output: 120

Task:

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

• Examine improvements in clarity and correctness.

Output:



```python
#write a python program to print factorial of a number.
#Input: 5
#Output: 120
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Test the factorial function with multiple inputs
test_values = [5, 0, 1, 7, -3]
for val in test_values:
    print(f"Factorial of {val} is: {factorial(val)}")
# Fixed version of the factorial function with edge case handling
```

```
PS C:\Users\tejas\College\AI ASS Code> & C:/Users/tejas/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/tejas/College/AI ASS Code/lab3/Sec.py"
Factorial of 5 is: 120
Factorial of 0 is: 1
Factorial of 1 is: 1
Factorial of 7 is: 5040
Factorial of -3 is: Factorial is not defined for negative numbers
PS C:\Users\tejas\College\AI ASS Code> 
```

Anlysis:

One-shot code clearly handles the base case (0! = 1)

Zero-shot version misses explicit handling of zero

One-shot solution is easier to understand and more structured

One-shot result is mathematically more correct

Example helps the AI generate safer and clearer logic.

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

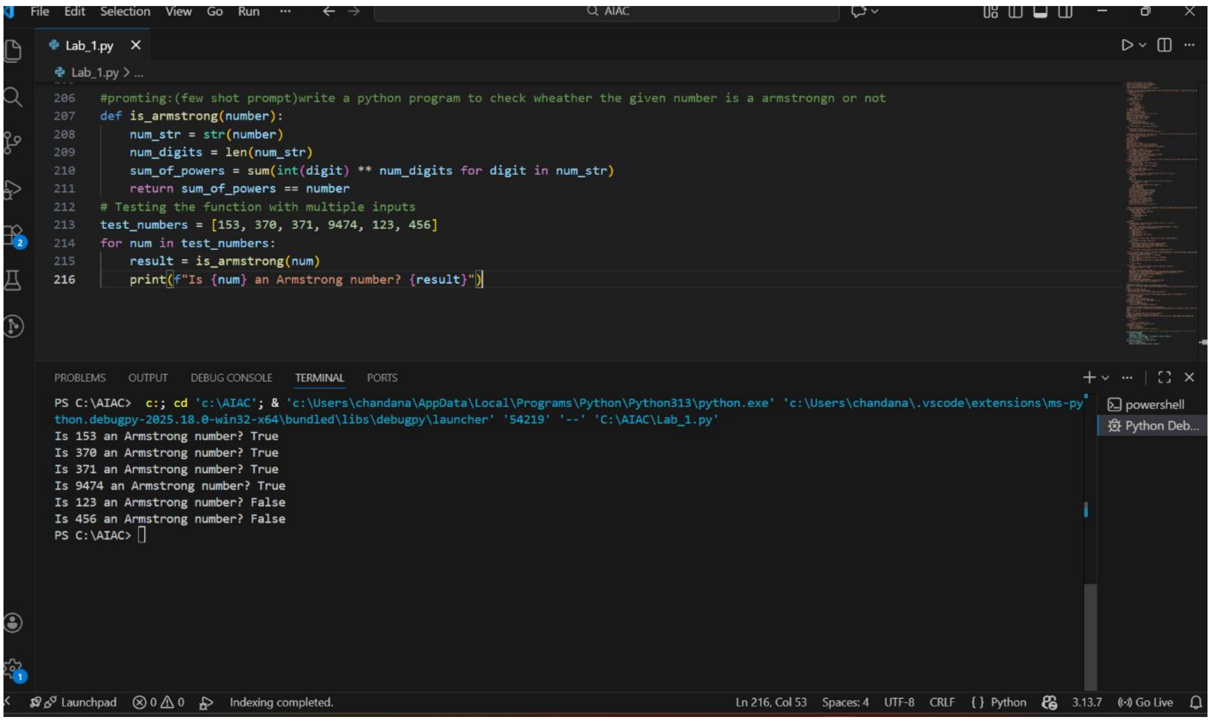
• Input: 370 → Output: Armstrong Number

• Input: 123 → Output: Not an Armstrong Number

Task:

• Analyze how multiple examples influence code structure and accuracy.

• Test the function with boundary values and invalid inputs.


File Edit Selection View Go Run ...
AIAC
Lab_1.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Launchpad
Indexing completed.
Ln 216, Col 53 Spaces: 4 UTF-8 CRLF {} Python 3.13.7 Go Live

```python
#promting:(few shot prompt)write a python program to check wheather the given number is a armstrongn or not
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
# Testing the function with multiple inputs
test_numbers = [153, 370, 371, 9474, 123, 456]
for num in test_numbers:
    result = is_armstrong(num)
    print(f"Is {num} an Armstrong number? {result}")
```

```
PS C:\AIAC>  c:; cd 'c:\AIAC'; & 'c:\Users\chandana\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\chandana\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54219' '--' 'C:\AIAC\Lab_1.py'
Is 153 an Armstrong number? True
Is 370 an Armstrong number? True
Is 371 an Armstrong number? True
Is 9474 an Armstrong number? True
Is 123 an Armstrong number? False
Is 456 an Armstrong number? False
PS C:\AIAC> 
```

Analysis:

Giving examples helps the AI understand what kind of answer is expected.

Multiple examples make the code cleaner and more logical.

Showing both correct and incorrect cases avoids confusion.

Testing small numbers like 0 and 1 ensures the function works properly.

Checking wrong inputs makes the program safer and more reliable.

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

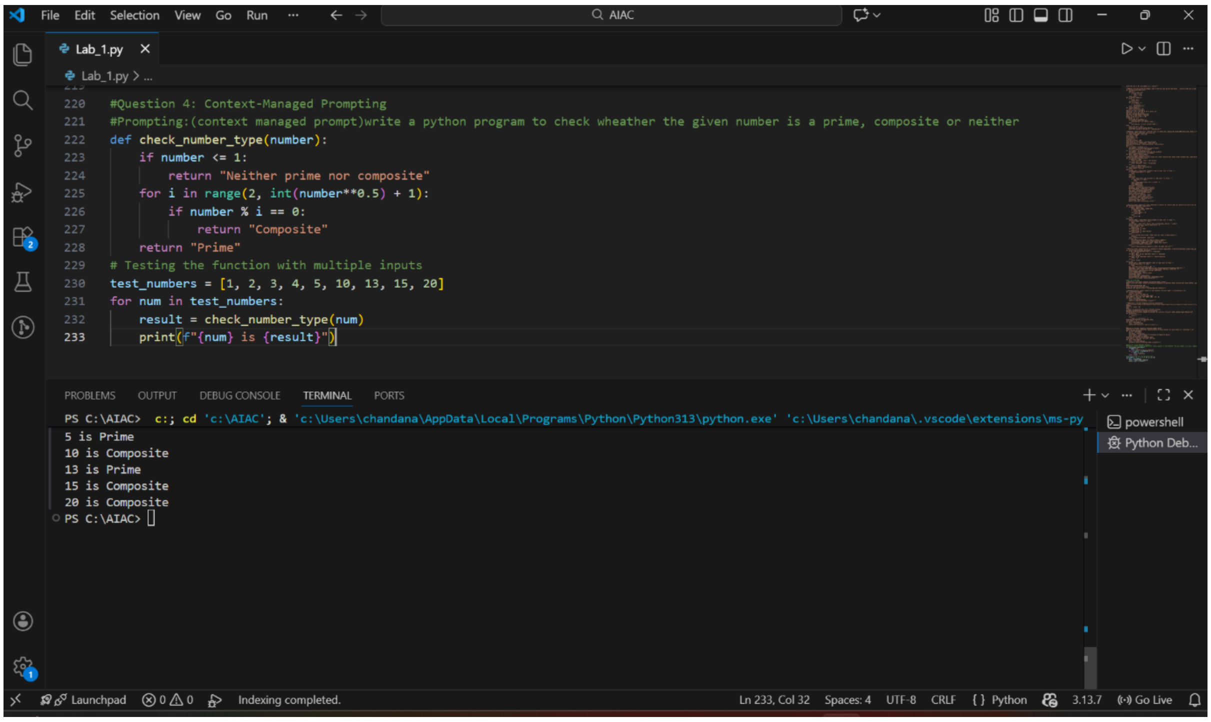
Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.


File Edit Selection View Go Run ···
AIAC
Lab_1.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Launchpad
Indexing completed.
Ln 233, Col 32 Spaces: 4 UTF-8 CRLF {} Python 3.13.7 Go Live

```python
#Question 4: Context-Managed Prompting
#Prompting:(context managed prompt)write a python program to check wheather the given number is a prime, composite or neither
def check_number_type(number):
    if number <= 1:
        return "Neither prime nor composite"
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return "Composite"
    return "Prime"
# Testing the function with multiple inputs
test_numbers = [1, 2, 3, 4, 5, 10, 13, 15, 20]
for num in test_numbers:
    result = check_number_type(num)
    print(f"{num} is {result}")
```

```
PS C:\AIAC>  c:; cd 'c:\AIAC'; & 'c:\Users\chandana\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\chandana\.vscode\extensions\ms-py
5 is Prime
10 is Composite
13 is Prime
15 is Composite
20 is Composite
PS C:\AIAC> 
```

Analysis:

Clear instructions help the AI understand exactly what is needed.

Input validation avoids crashes from wrong inputs.

Efficient logic makes the program faster and smarter.

The AI performs better than with simple prompts.

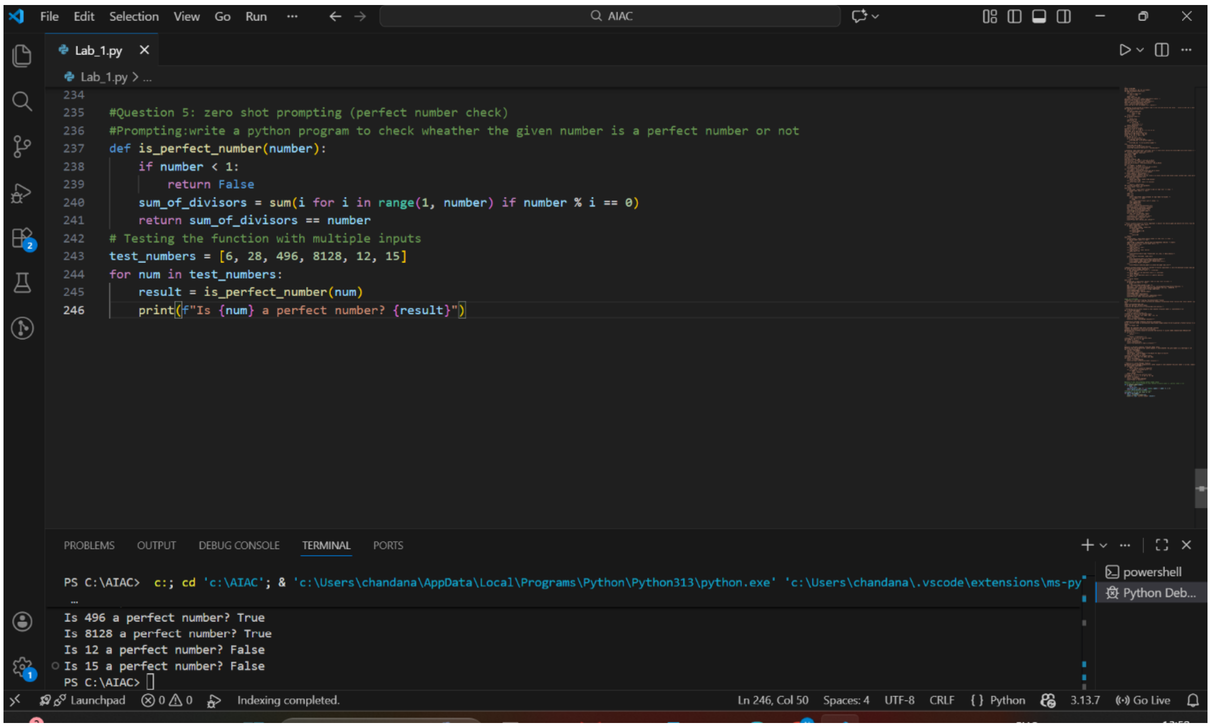
Results are clearer and more reliable.

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.



```python
#Question 5: zero shot prompting (perfect number check)
#Prompting:write a python program to check wheather the given number is a perfect number or not
def is_perfect_number(number):
    if number < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
    return sum_of_divisors == number
# Testing the function with multiple inputs
test_numbers = [6, 28, 496, 8128, 12, 15]
for num in test_numbers:
    result = is_perfect_number(num)
    print(f"Is {num} a perfect number? {result}")
```

```
PS C:\AIAC>  c:; cd 'c:\AIAC'; & 'c:\Users\chandana\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\chandana\.vscode\extensions\ms-py
 ...
Is 496 a perfect number? True
Is 8128 a perfect number? True
Is 12 a perfect number? False
Is 15 a perfect number? False
PS C:\AIAC>
```

Analysis:

The AI works only with instructions, no examples.

The code usually works but may miss some cases.

Testing with different numbers shows if it’s correct.

The logic may be slower without optimization.

Extra checks improve accuracy.

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

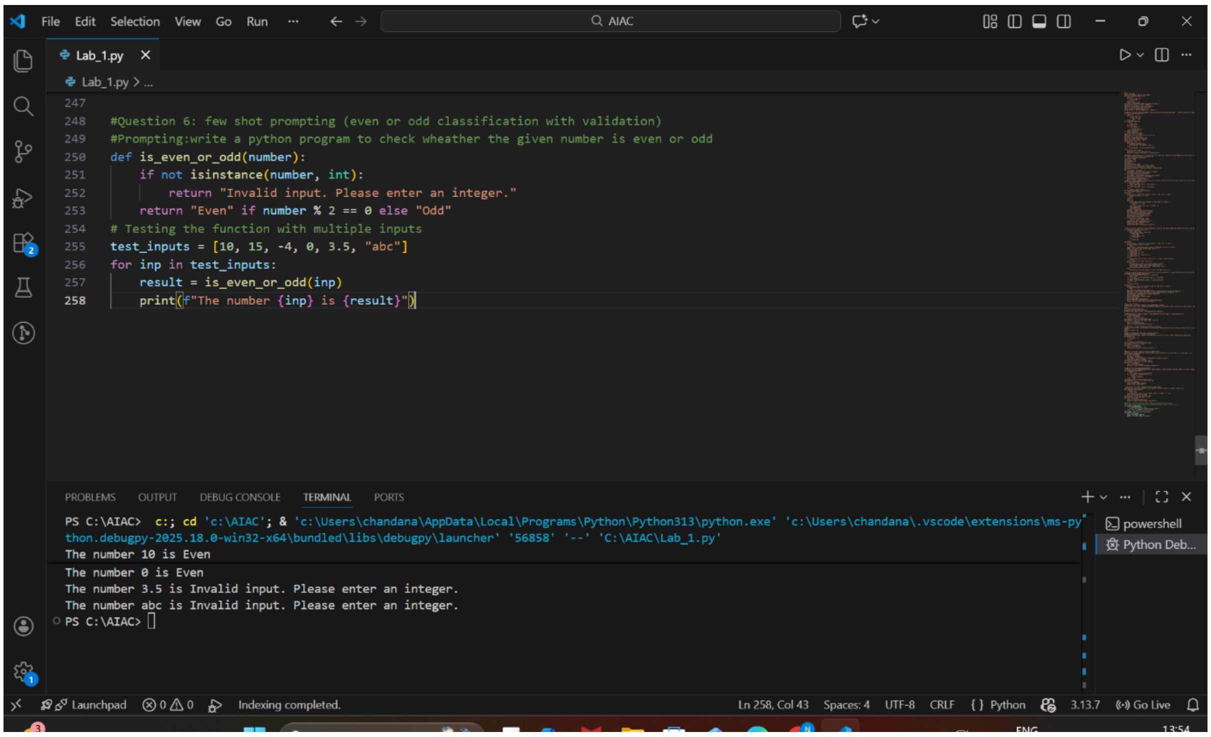
• Input: 15 → Output: Odd

• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output clarity.

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



```python

#Question 6: few shot prompting (even or odd classification with validation)
#Prompting:write a python program to check wheather the given number is even or odd
def is_even_or_odd(number):
    if not isinstance(number, int):
        return "Invalid input. Please enter an integer."
    return "Even" if number % 2 == 0 else "Odd"
# Testing the function with multiple inputs
test_inputs = [10, 15, -4, 0, 3.5, "abc"]
for inp in test_inputs:
    result = is_even_or_odd(inp)
    print(f"The number {inp} is {result}")
```

```
PS C:\AIAC>  c:; cd 'c:\AIAC'; & 'c:\Users\chandana\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\chandana\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56858' '--' 'C:\AIAC\Lab_1.py'
The number 10 is Even
The number 0 is Even
The number 3.5 is Invalid input. Please enter an integer.
The number abc is Invalid input. Please enter an integer.
PS C:\AIAC> 
```

File Edit Selection View Go Run
AIAC
Lab_1.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Launchpad
Indexing completed.
Ln 258, Col 43 Spaces: 4 UTF-8 CRLF { } Python 3.13.7 Go Live

Analysis:

Examples make the task easy to understand.

The AI gives clear even or odd results.

Input validation improves with examples.

Negative numbers are handled properly.

Wrong inputs are easier to detect.