

AI Assisted Coding

Week2 – Monday

Name: Lalu prasad Aroori

Batch: 07

Roll No.: 2303A51948

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.
- Test the code with multiple inputs.
- Identify any logical errors or missing edge-case handling.

Prompt:

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

The function should return True if the number is a palindrome and False otherwise.

```

1  #Write a Python function that checks whether a given number is a palindrome.
2  #The function should return True if the number is a palindrome and False otherwise.
3
4  # Function to check if a number is a palindrome:
5  def is_palindrome(number):
6      # Convert the number to string
7      str_num = str(number)
8
9      # Check if the string is the same forwards and backwards
10     return str_num == str_num[::-1]
11
12     # Example usage:
13     print(is_palindrome(121)) # True
14     print(is_palindrome(-121)) # False
15     print(is_palindrome(10)) # False
16

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python Debug Console

```

PS D:\3_2_SEM\AI_assisted_coding> & 'c:\Users\Lalu prasad Aroori\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56067' '--' 'D:\3_2_SEM\AI_assisted_coding\Zero-Shot Prompting.py'
True
False
False
PS D:\3_2_SEM\AI_assisted_coding>

```

Remarks

- The code correctly identifies palindrome numbers.
- Limitation:** Does not handle negative numbers (e.g., -121).
- Zero-shot prompting works but may miss edge-case handling.
- Logic is clear but could be improved with input validation.

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.

Prompt:

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

Example:

Input: 5

Output: 120

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like 'Welcome', 'Zero-Shot Prompting.py', and 'One-Shot Prompting.py'.
- Editor:** Displays a Python script named 'One-Shot Prompting.py' containing code to calculate the factorial of a number. The code includes examples and handles negative numbers.
- Terminal:** Shows the command line output of running the script, which prints 120 for input 5.
- Status Bar:** Shows the current file is 'One-Shot Prompting.py', and other details like line count (Ln 25), column (Col 1), and encoding (UTF-8).

Remarks

- One-shot prompting improved clarity and correctness.
- Handles negative numbers properly.
- Compared to zero-shot, the logic is more structured.
- Demonstrates how **examples guide AI behavior**.

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.

Prompt:

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

Examples:

Input: 153 → Output: Armstrong Number

Input: 370 → Output: Armstrong Number

Input: 123 → Output: Not an Armstrong Number

The Python Debug Console shows the output of running the script with the provided examples, confirming the results as True for 153 and 370, and False for 123.

Remarks

- Few-shot prompting resulted in clean and accurate logic.
- Works for boundary values like 0.
- Code readability and correctness improved.
- Demonstrates the effectiveness of multiple examples.

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.

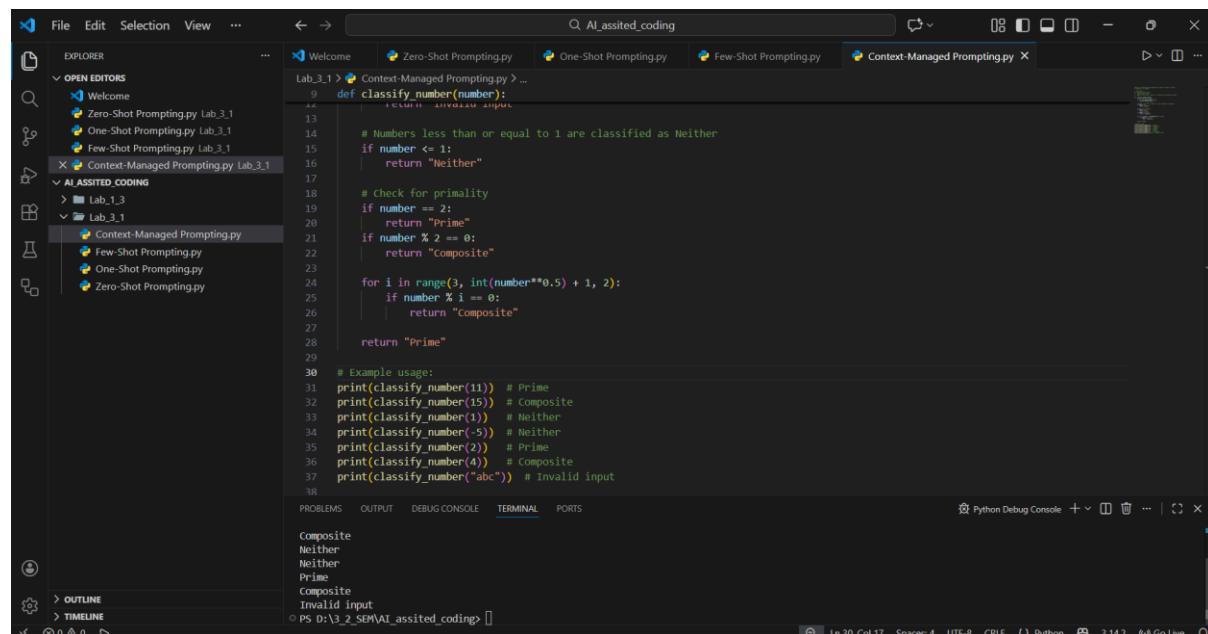
Prompt:

#Write an optimized Python program to classify a number as Prime,

Composite, or Neither.

Constraints:

- Handle invalid inputs
- Use efficient logic
- Numbers less than or equal to 1 should be classified as Neither



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the 'AI_ASSISTED_CODING' folder, including 'Context-Managed Prompting.py' which is the active file.
- Code Editor:** Displays the following Python code:

```
def classify_number(number):  
    if number <= 1:  
        return "Neither"  
  
    for i in range(2, int(number**0.5) + 1, 2):  
        if number % i == 0:  
            return "Composite"  
  
    return "Prime"  
  
# Example usage:  
print(classify_number(11)) # Prime  
print(classify_number(15)) # Composite  
print(classify_number(1)) # Neither  
print(classify_number(.5)) # Neither  
print(classify_number(2)) # Prime  
print(classify_number(4)) # Composite  
print(classify_number("abc")) # Invalid input
```

- Terminal:** Shows the output of the code execution:

```
Composite  
Neither  
Neither  
Prime  
Composite  
Invalid input
```
- Status Bar:** Shows the current file path as 'D:\3_2_SEMP\AI_assisted_coding> []', line 30, column 17, and other standard status bar information.

Remarks

- Context-managed prompting produces optimized and robust code.
- Efficient logic using square-root optimization.
- Proper input validation included.
- Best quality among all prompting techniques.

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

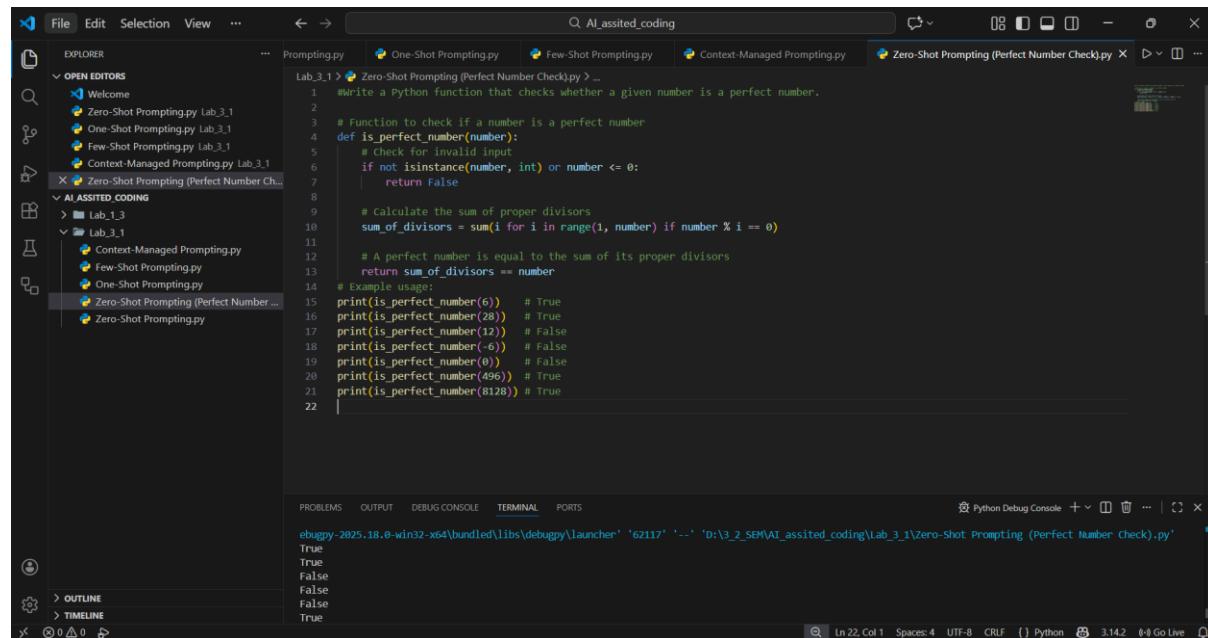
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

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 number is a perfect number.



```
Lab_3_1 > Zero-Shot Prompting (Perfect Number Check).py ...
1  #write a Python function that checks whether a given number is a perfect number.
2
3  # Function to check if a number is a perfect number
4  def is_perfect_number(number):
5      # Check for invalid input
6      if not isinstance(number, int) or number <= 0:
7          return False
8
9      # calculate the sum of proper divisors
10     sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
11
12     # A perfect number is equal to the sum of its proper divisors
13     return sum_of_divisors == number
14
15 print(is_perfect_number(6))  # True
16 print(is_perfect_number(8))  # True
17 print(is_perfect_number(12)) # False
18 print(is_perfect_number(6))  # False
19 print(is_perfect_number(0))  # False
20 print(is_perfect_number(496)) # True
21 print(is_perfect_number(8128)) # True
22
```

Python Debug Console

```
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "62117" "-"- "D:\3_2_SEM\AI_assisted_coding\Lab_3_1\Zero-Shot Prompting (Perfect Number Check).py"
True
True
False
False
True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Ln 22, Col 1 Spaces:4 UTF-8 CRLF {} Python 3.14.2 Go Live

Remarks

- Works correctly for perfect numbers.
- **Inefficient** for large numbers (checks all values till n).
- Zero-shot lacks optimization like checking till $n/2$.
- Improvement possible with better constraints.

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 whether a number is Even or Odd with proper input validation.

Examples:

Input: 8 → Output: Even

Input: 15 → Output: Odd

Input: 0 → Output: Even

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view of files under "AI ASSISTED CODING". Opened files include "Few-Shot Prompting.py", "Context-Managed Prompting.py", "Zero-Shot Prompting (Perfect Number Check).py", and "Few-Shot Prompting (Even or Odd with Validation).py".
- Editor:** The main editor area displays a Python script named "Few-Shot Prompting (Even or Odd with Validation).py". The code defines a function `check_even_odd` that checks if a number is even or odd, handling invalid inputs.
- Terminal:** The terminal tab shows the output of running the script with various inputs:

```
Even
Odd
Even
Even
Invalid input
Invalid input
Odd
```
- Status Bar:** Shows the current file is "Few-Shot Prompting (Even or Odd with Validation).py", line 23, column 35, with 4 spaces, encoding as UTF-8, and Python 3.14.2.

Remarks

- Few-shot prompting significantly improved input validation.
- Handles negative numbers and non-integers correctly.
- Output clarity is high.
- Demonstrates strong influence of examples on AI output.