

AI ASSISTED CODING

M.Rohith Reddy
2303A51111

BATCH - 03

23 - 01 - 2026

ASSIGNMENT – 3.5

LAB-03:

TASK-01: Zero-Shot Prompting – Leap Year Check

- Record the AI-generated code.
- Test with years like 1900, 2000, 2024.
- Identify logical flaws or missing conditions.

Prompt: Write a Python function to check whether a given year is a leap year

Code:

The screenshot shows a code editor interface with a dark theme. On the left is a file tree containing several Python files: factorial using recursion.py, lab_assignment_1_5.py, lab_assignment_2_2.py, lab_assignment_3_2.py, lab_assignment_3_5.py (which is the active file), and rough.py. The main workspace displays the following Python code:

```
#Write a Python function to check whether a given year is a Leap year
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    return False
print(is_leap_year(1900))
print(is_leap_year(2000))
print(is_leap_year(2024))
```

Below the code, the terminal window shows the execution of the script:

```
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
False
True
True
PS D:\AI Assisted Coding>
```

The bottom status bar indicates the file is a Python file (Python), version 3.11.4, and the code is valid (Go Live).

Analysis

- The code incorrectly marks 1900 as a leap year.
 - Missing century-year rule (divisible by 100 but not by 400).

TASK-02: One-Shot Prompting (GCD of Two Numbers)

- Compare with a zero-shot solution.
 - Analyse algorithm efficiency.

Prompt: Write a Python function to find the GCD of two numbers.

Example: Input: 12, 18 → Output: 6

CODE:

The screenshot shows the Visual Studio Code interface with the 'AI ASSISTED CODING' extension active. The left sidebar displays a tree view of files: factorial using recursion.py, lab_assignment_1_5.py, lab_assignment_2_2.py, lab_assignment_3_2.py, lab_assignment_3_5.py (selected), and rough.py. The main editor area contains a Python script for calculating the Greatest Common Divisor (GCD) using recursion:

```
#Write a Python function to find GCD of two numbers. Example: Input: 12, 18 → Output: 6
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
print(gcd(12, 18)) # Output: 6
```

The bottom navigation bar includes tabs for PROBLEMS, DEBUG CONSOLE, OUTPUT, TERMINAL, PORTS, SQL HISTORY, and TASK MONITOR. The TERMINAL tab is selected, showing the command line output of the script execution:

```
AI Assisted Coding/lab_assignment_3_5.py"
False
True
True
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
AI Assisted Coding/lab_assignment_3_5.py"
6
PS D:\AI Assisted Coding> []
```

Zero-Shot Comparison

Zero-shot solutions often use brute-force loops, whereas this uses the efficient Euclidean algorithm.

Efficiency Analysis

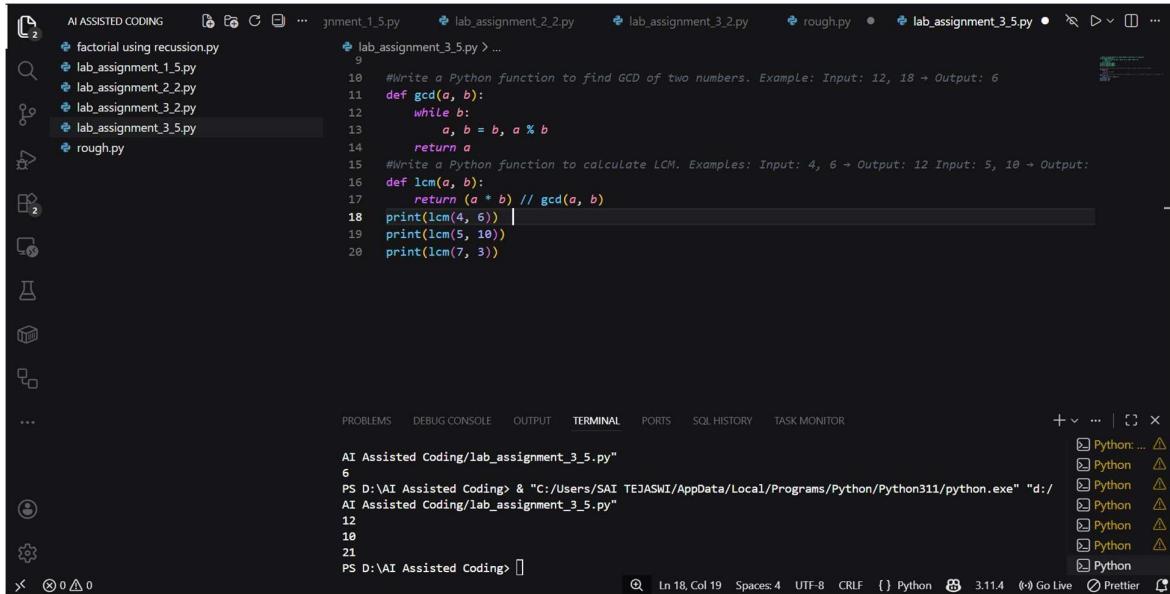
- Time Complexity: $O(\log \min(a, b))$
 - More efficient than trial division.

TASK-03: Few-Shot Prompting (LCM Calculation)

- Examine how examples guide formula selection.
- Test edge cases.

Prompt: Write a Python function to calculate LCM. Examples: Input: 4, 6 → Output: 12 Input: 5, 10 → Output: 10 Input: 7, 3 → Output: 21

CODE:



```
AI ASSISTED CODING ... jnment_1_5.py lab_assignment_2_2.py lab_assignment_3_2.py rough.py ... lab_assignment_3_5.py ... factorial using recursion.py lab_assignment_1_5.py > ... 9 10 #Write a Python function to find GCD of two numbers. Example: Input: 12, 18 → Output: 6 11 def gcd(a, b): 12     while b: 13         a, b = b, a % b 14     return a 15 #Write a Python function to calculate LCM. Examples: Input: 4, 6 → Output: 12 Input: 5, 10 → Output: 10 16 def lcm(a, b): 17     return (a * b) // gcd(a, b) 18 print(lcm(4, 6)) | 19 print(lcm(5, 10)) 20 print(lcm(7, 3))
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS SQL HISTORY TASK MONITOR

AI Assisted Coding/lab_assignment_3_5.py" 6 PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py" 12 18 21 PS D:\AI Assisted Coding> []

Ln 18, Col 19 Spaces: 4 UTF-8 CRLF {} Python 3.11.4 Go Live Prettier

Observation

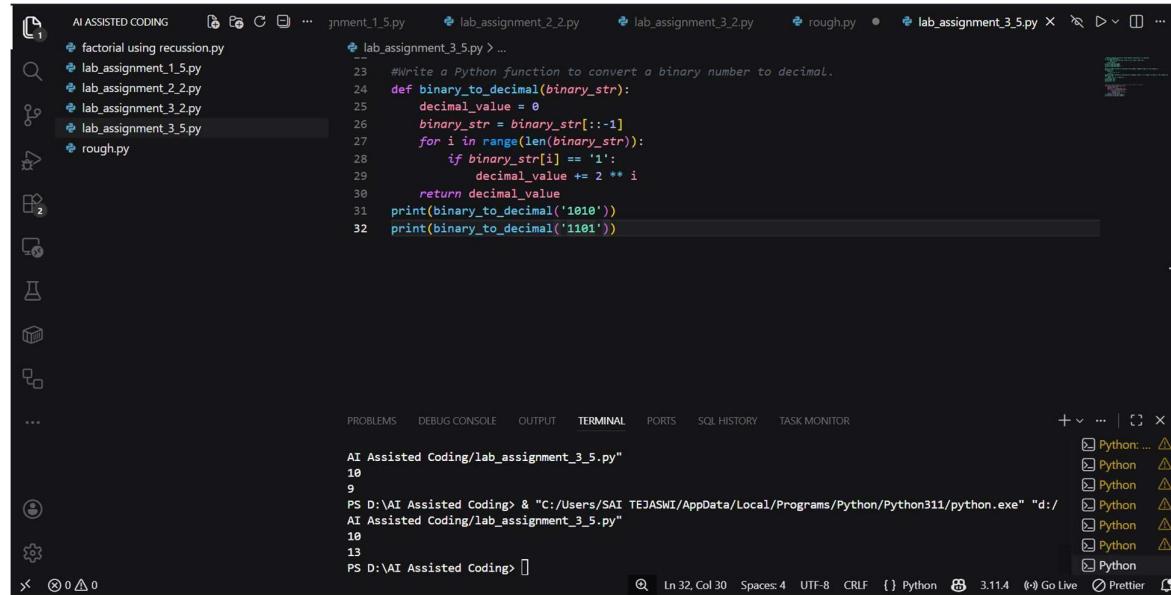
- Formula-based approach chosen due to examples.
- Needs handling when either input is zero.

TASK-04: Zero-Shot Prompting – Binary to Decimal Conversion

- Test with valid and invalid binary inputs.
- Identify missing validation logic.

PROMPT: Write a Python function to convert a binary number to decimal.

CODE:



A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with several Python files: factorial using recursion.py, lab_assignment_1_5.py, lab_assignment_2_2.py, lab_assignment_3_2.py, lab_assignment_3_5.py, rough.py, and lab_assignment_3_5.py (the active file). The main editor area contains the following Python code:

```
23  #Write a Python function to convert a binary number to decimal.
24  def binary_to_decimal(binary_str):
25      decimal_value = 0
26      binary_str = binary_str[::-1]
27      for i in range(len(binary_str)):
28          if binary_str[i] == '1':
29              decimal_value += 2 ** i
30      return decimal_value
31 print(binary_to_decimal('1010'))
32 print(binary_to_decimal('1101'))
```

The terminal at the bottom shows the command being run and its output:

```
AI Assisted Coding/lab_assignment_3_5.py"
10
9
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
10
13
```

The status bar at the bottom indicates the file is 3.11.4, Python, and Prettier are active.

Analysis

- No validation for invalid binary digits.
- Error occurs for invalid input.

TASK-05: One-Shot Prompting – Decimal to Binary conversion

- Compare clarity with zero-shot output.
- Analyze handling of zero and negative numbers.

PROMPT: Write a Python function to convert decimal to binary. Example: Input:

10 → Output: 1010

CODE:

The screenshot shows a dark-themed instance of Visual Studio Code. In the center is a code editor with Python code. The code defines a function `decimal_to_binary` that converts a decimal number to binary. It includes a docstring, a check for zero, and a loop that repeatedly divides the number by 2 and prepends the remainder to the result string. A call to `print(decimal_to_binary(10))` is at the bottom. Below the code editor is a terminal window showing the command line and the output of running the script. The terminal shows the command `python lab_assignment_3_5.py`, the output `1010`, and the prompt `PS D:\AI Assisted Coding>`. The bottom status bar indicates the file is 43 lines long, has 29 spaces, and is in UTF-8 encoding.

```
#Write a Python function to convert decimal to binary. Example: Input: 10 → Output: 1010
def decimal_to_binary(n):
    if n == 0:
        return '0'
    binary_str = ''
    while n > 0:
        binary_str = str(n % 2) + binary_str
        n //= 2
    return binary_str
print(decimal_to_binary(10))|
```

```
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
10
13
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
1010
PS D:\AI Assisted Coding>
```

Analysis

- Correct for positive numbers.
- For 0 returns '0'.
- Negative numbers return prefixed binary, needs clarification.

TASK-06: Few-Shot Prompting – Harshad Number Check

- Test boundary conditions.
- Evaluate robustness

PROMPT: Write a Python function to check Harshad Number. Examples: Input: 18

→ Output: Harshad Number Input: 21 → Output: Harshad Number Input: 19 →

Output: Not a Harshad Number

CODE:

The screenshot shows the VS Code interface with the "AI ASSISTED CODING" tab selected. The left sidebar displays files: factorial using recursion.py, lab_assignment_1_5.py, lab_assignment_2_2.py, lab_assignment_3_2.py, lab_assignment_3_5.py, and rough.py. The main editor area contains Python code for checking Harshad numbers:

```
45  #Write a Python function to check Harshad Number. Examples: Input: 18 -> Output: Harshad Number Input: 19 -> Output: Not a Harshad Number
46  def is_harshad_number(n):
47      digit_sum = sum(int(digit) for digit in str(n))
48      if n % digit_sum == 0:
49          return "Harshad Number"
50      else:
51          return "Not a Harshad Number"
52
53 print(is_harshad_number(18))
54 print(is_harshad_number(21))
55 print(is_harshad_number(19))
```

The terminal below shows the execution of the code:

```
AI Assisted Coding/lab_assignment_3_5.py
1010
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_3_5.py"
Harshad Number
Harshad Number
Not a Harshad Number
PS D:\AI Assisted Coding>
```

Analysis

- Works correctly for positive integers.
- Division by zero occurs when input is 0.