

AI ASSISTED CODING

NASANI SAI PAVAN

2303A51117

BATCH – 03

23 – 01 – 2026

ASSIGNMENT – 3.5

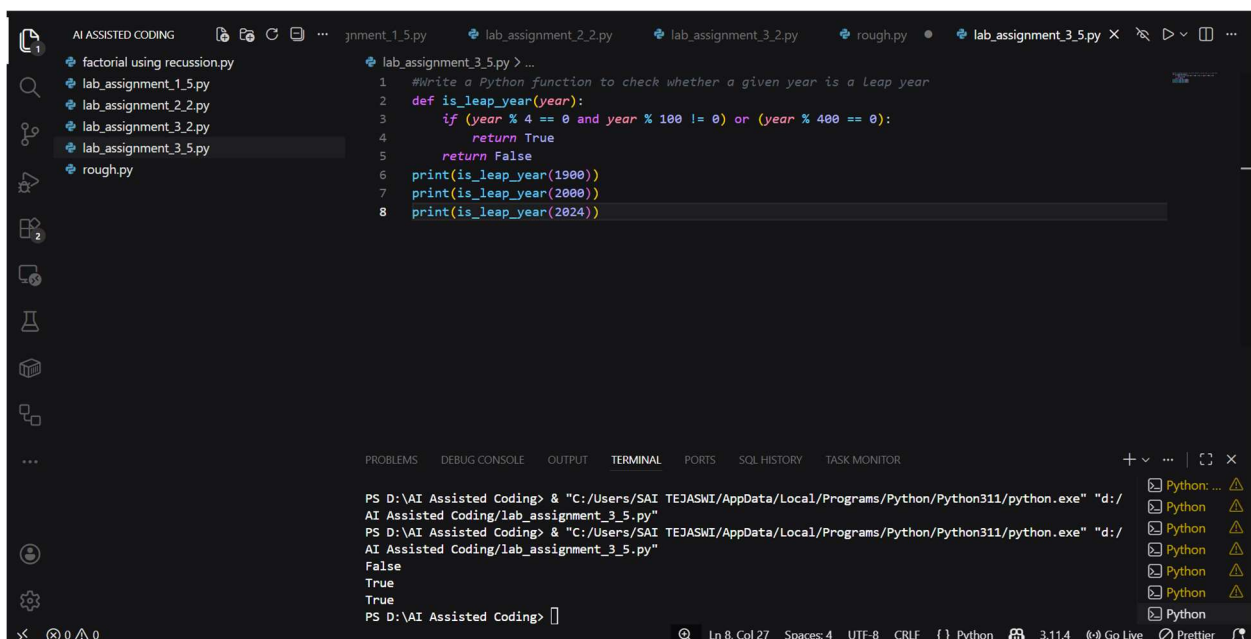
LAB-03:

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:



```
AI ASSISTED CODING
lab_assignment_1_5.py
lab_assignment_2_2.py
lab_assignment_3_2.py
lab_assignment_3_5.py
rough.py
lab_assignment_3_5.py > ...

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

PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS  SQL HISTORY  TASK MONITOR

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
True
PS D:\AI Assisted Coding>

Ln 8, Col 27  Spaces: 4  UTF-8  CRLF  {} Python  3.11.4  Go Live  Prettier
```

Analysis

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

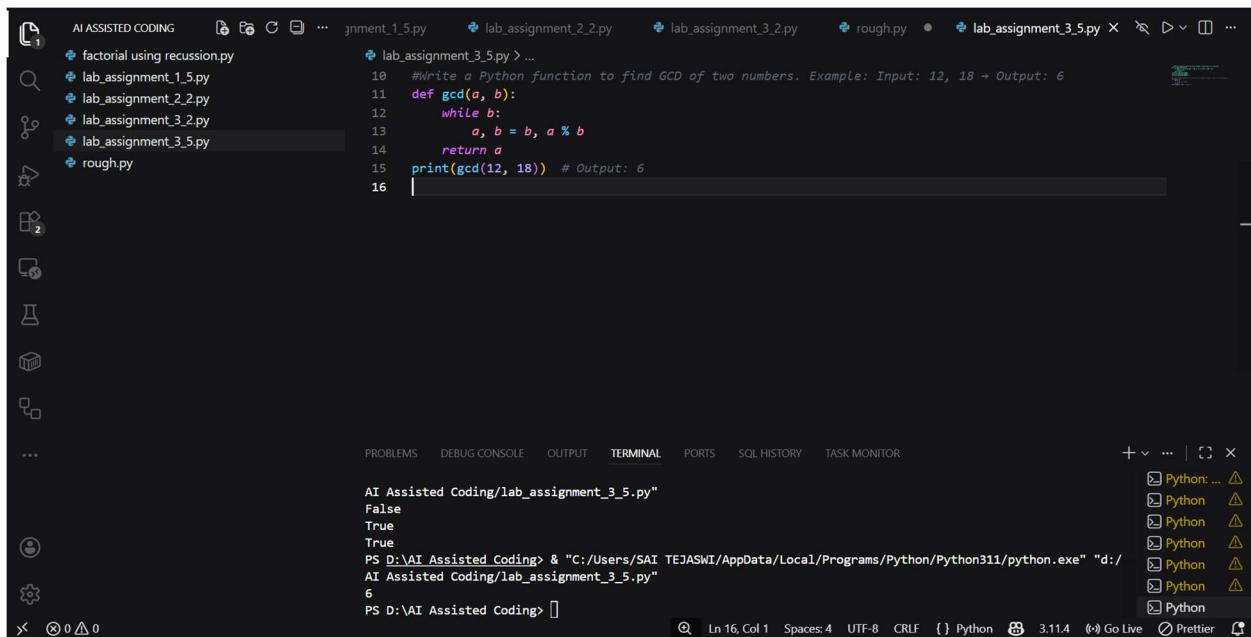
TASK-02: One-Shot Prompting (GCD of Two Numbers)

- 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 a code editor with a file explorer on the left containing files like `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 displays a Python function `gcd(a, b)` using a `while` loop to implement the Euclidean algorithm. The function is called with `gcd(12, 18)` and the output `6` is printed. The bottom panel shows the terminal output of the command `python lab_assignment_3_5.py`, which prints `False`, `True`, `True`, and `6`.

```
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 print(gcd(12, 18)) # Output: 6
16
```

```
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"
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

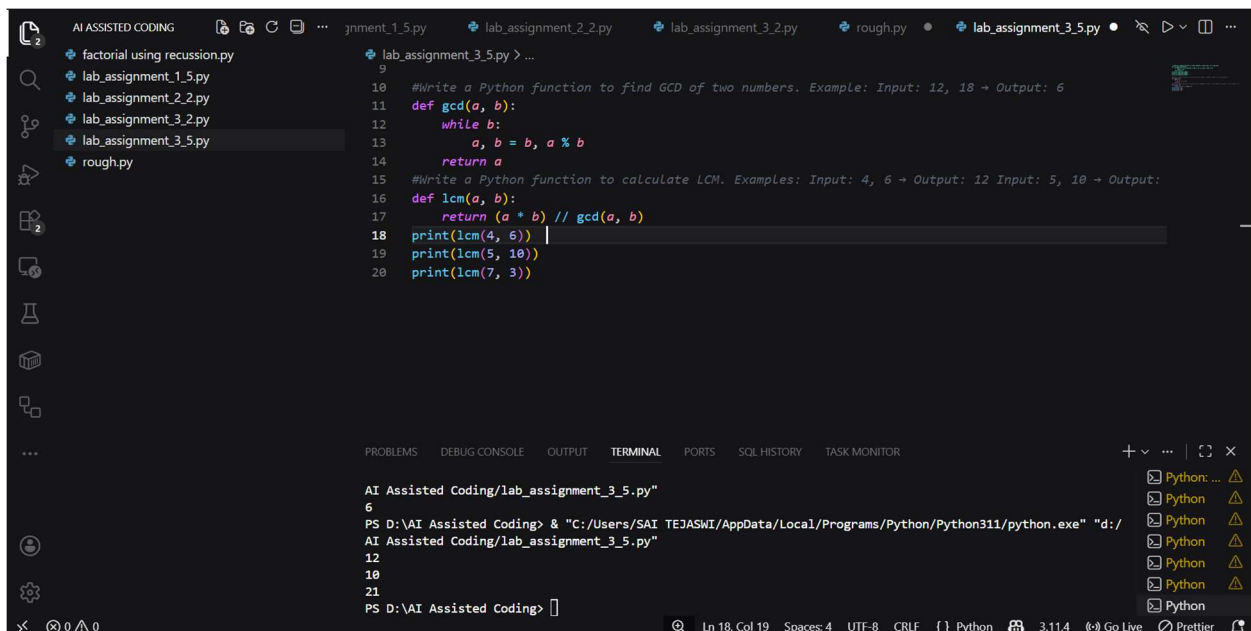
- Time Complexity: $O(\log \min(a, b))$
- More efficient than trial division.

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
lab_assignment_1_5.py
lab_assignment_2_2.py
lab_assignment_3_2.py
lab_assignment_3_5.py
rough.py

lab_assignment_3_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
10
21
PS D:\AI Assisted Coding> |
Ln 18, Col 19 Spaces: 4 UTF-8 CRLF {} Python 3.11.4 Go Live Prettier
```

Observation

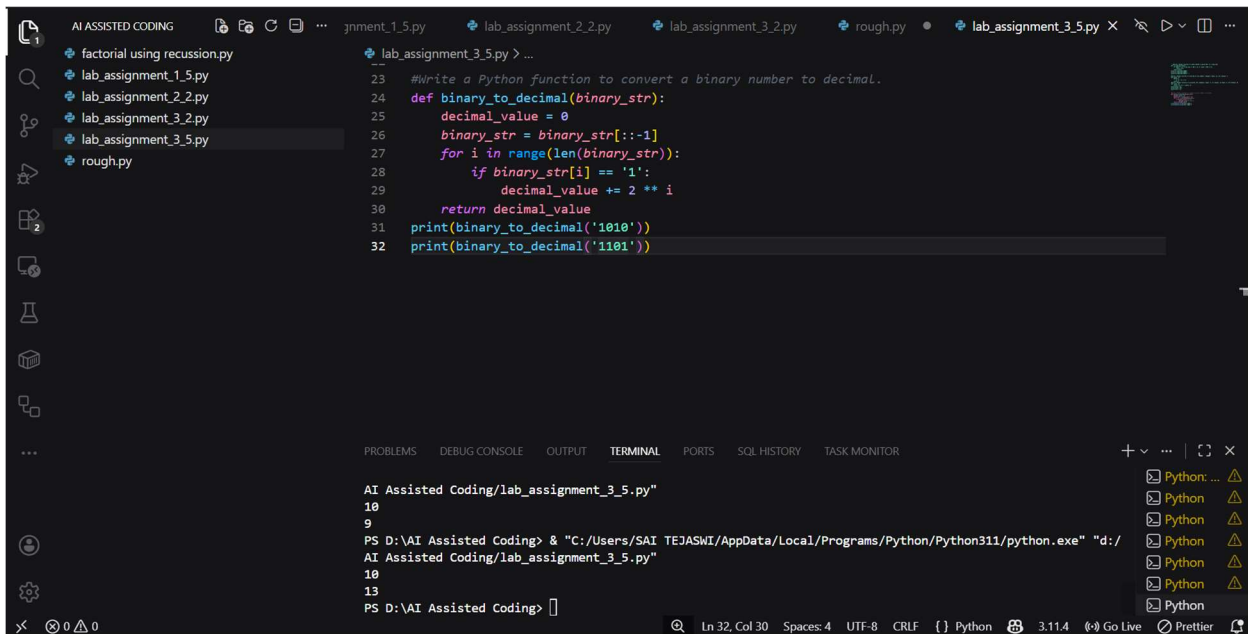
- Formula-based approach chosen due to examples.
- Needs handling when either input is zero.

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:



```
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'))
```

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
PS D:\AI Assisted Coding>

Analysis

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

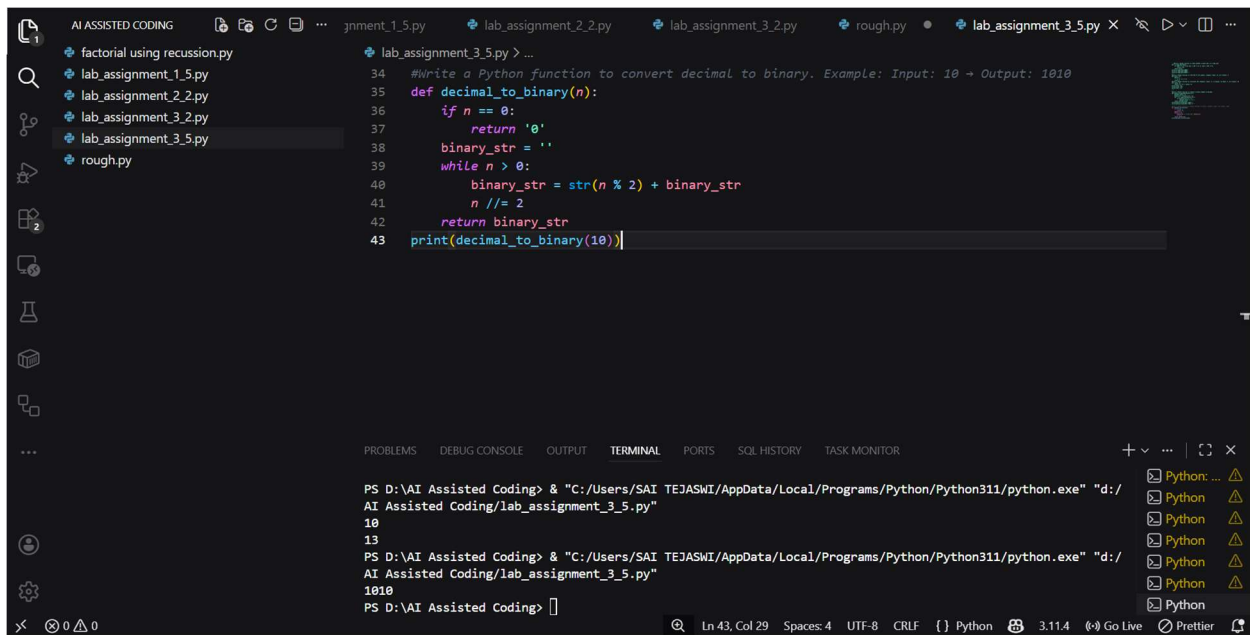
TASK-05: One-Shot Prompting – Decimal to Binary conversion

- 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:



```
34 #Write a Python function to convert decimal to binary. Example: Input: 10 → Output: 1010
35 def decimal_to_binary(n):
36     if n == 0:
37         return '0'
38     binary_str = ''
39     while n > 0:
40         binary_str = str(n % 2) + binary_str
41         n //= 2
42     return binary_str
43 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 image shows a Visual Studio Code editor window with a Python file named `lab_assignment_3_5.py` open. The code defines a function `is_harshad_number(n)` that checks if a number is a Harshad number by dividing it by the sum of its digits. The function returns "Harshad Number" if the division is exact, and "Not a Harshad Number" otherwise. The script tests the function with inputs 18, 21, and 19.

```
45 #Write a Python function to check Harshad Number. Examples: Input: 18 → Output: Harshad Number Input:
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 print(is_harshad_number(18))
53 print(is_harshad_number(21))
54 print(is_harshad_number(19))
```

The terminal output shows the execution of the script, confirming that 18 and 21 are Harshad numbers, while 19 is not.

```
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.