AI ASSISTED CODING

ASSIGNMENT 3.5

Name: G.V.N Hasini

HtNo: 2303A51099

Question 1: Zero-Shot Prompting (Leap Year Check)

Write a zero-shot prompt to generate a Python function that checks whether a given year is a leap year.

Week2 -

Task:

• Record the AI-generated code.

• Test with years like 1900, 2000, 2024.

• Identify logical flaws or missing conditions.

```python
def is_leap_year(year):

    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return f"{year} is a leap year"
    else:
        return f"{year} is not a leap year"


# Test the function
year = int(input("Enter a year: "))
print(is_leap_year(year))
```

```
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter a year: 1900
1900 is not a leap year
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter a year: 2000
2000 is a leap year
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter a year: 2024
2024 is a leap year
```

The program does not check for invalid year values such as 0 or negative numbers. It only validates non-numeric input and assumes any integer year is valid. There is no explicit condition to restrict the year to a realistic range. Apart from these missing checks, the leap year logic itself has no flaws.

Question 2: One-Shot Prompting (GCD of Two Numbers)

Write a one-shot prompt with one example to generate a Python

function that finds the Greatest Common Divisor (GCD) of two numbers.

Example:

Input: 12, 18 → Output: 6

Task:

• Compare with a zero-shot solution.

• Analyze algorithm efficiency.

```
a=12
b=18
display GCD is 6
user input a and b
'''
a = int(input("Enter first number (a): "))
b = int(input("Enter second number (b): "))
def compute_gcd(x, y):
    while(y):
        x, y = y, x % y
    return x
gcd = compute_gcd(a, b)
print("GCD is", gcd)
```

```
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter first number (a): 12
Enter second number (b): 18
GCD is 6
PS C:\Users\hasin\OneDrive\Desktop\java programming>
```

```
# Generate python code that finds greatest common divisor (GCD) of two numbers user inputs of a and b
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
# Get user input
a = int(input("Enter first number (a): "))
b = int(input("Enter second number (b): "))
# Calculate GCD
result = gcd(a, b)
# Print the result
print(f"The greatest common divisor (GCD) of {a} and {b} is: {result}")
```

```
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter first number (a): 12
Enter second number (b): 18
The greatest common divisor (GCD) of 12 and 18 is: 6
PS C:\Users\hasin\OneDrive\Desktop\java programming>
```

Both solutions correctly compute the GCD using **the Euclidean algorithm,** which is an efficient and standard approach. The zero-shot solution is more straightforward and concise, while the one shot solution is better structured and reusable. In terms of efficiency, both have the same time complexity of O(log min(a, b)), which is very fast even for large numbers. There is no performance difference between them; the main difference is code organization and reusability, not logic or efficiency.

Question 3: Few-Shot Prompting (LCM Calculation)

Write a few-shot prompt with multiple examples to generate a Python function that computes the Least Common Multiple (LCM).

Examples:

• Input: 4, 6 → Output: 12

• Input: 5, 10 → Output: 10

• Input: 7, 3 → Output: 21

Task:

• Examine how examples guide formula selection.

• Test edge cases.

```
a=4
b=6
display LCM is 12
a=5
b=10
display LCM is 10
a=0
b=0
display invalid input
a='abc
b=5
display invalid input
give user input for a and b
'''
def compute_lcm(a, b):
    if a <= 0 or b <= 0:
        return "invalid input"
    greater = max(a, b)
    while True:
        if greater % a == 0 and greater % b == 0:
            return f"LCM is {greater}"
        greater += 1
try:
    a = int(input("Enter first number (a): "))
    b = int(input("Enter second number (b): "))
    result = compute_lcm(a, b)
    print(result)
```

```
invalid input
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desk
gnment 3"
Enter first number (a): 4
Enter second number (b): 6
LCM is 12
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desk
gnment 3"
Enter first number (a): abc
invalid input
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desk
gnment 3"
Enter first number (a): 2
Enter second number (b): abc
invalid input
PS C:\Users\hasin\OneDrive\Desktop\java programming>
```

Question 4: Zero-Shot Prompting (Binary to Decimal Conversion) Write a zero-shot prompt to generate a Python function that converts a binary number to decimal.

Task:

• Test with valid and invalid binary inputs.

• Identify missing validation logic.

```python
# generate a python function that converts a binary number to decimal number user input
def binary_to_decimal(binary_str):
    try:
        decimal_number = int(binary_str, 2)
        return decimal_number
    except ValueError:
        return "Invalid binary number"
# Example usage
user_input = input("Enter a binary number: ")
result = binary_to_decimal(user_input)
print(f"The decimal equivalent is: {result}")
```

```
Enter a binary number: ml
The decimal equivalent is: Invalid binary number
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assi
gnment 3"
Enter a binary number: 2.5
The decimal equivalent is: Invalid binary number
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assi
gnment 3"
Enter a binary number: 11111
The decimal equivalent is: 31
```

The code does not check for empty input, so an empty string would be treated as valid and return 0. It also does not handle leading/trailing spaces explicitly before processing. Negative binary values (like -101) are not considered and should be rejected clearly. Apart from these missing validations, the binary digit check (0 and 1) is correct.

Question 5: One-Shot Prompting (Decimal to Binary Conversion)

Write a one-shot prompt with an example to generate a Python function

that converts a decimal number to binary.

```
10
display 1010
'''
def decimal_to_binary(n):
    try:
        n=int(n)
    except ValueError:
        return "Invalid Input"
    if n<0:
        return "Invalid Input"
    if n==0:
        return "0"
    binary_str=""
    while n>0:
        binary_str=str(n%2)+binary_str
        n=n//2
    return binary_str
n=input("Enter a decimal number: ")
print(decimal_to_binary(n))
```

Example:

Input: 10 → Output: 1010

Task:

• Compare clarity with zero-shot output.

• Analyze handling of zero and negative numbers.

```
# generate a python code that converts a decimal number into binary number user input
n=int(input("Enter a decimal number: "))
binary_number = bin(n).replace("0b", "")
print("The binary representation of", n, "is", binary_number)
```

```
Enter a decimal number: 0
The binary representation of 0 is 0
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter a decimal number: -90
The binary representation of -90 is -1011010
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assignment 3"
Enter a decimal number: 2
The binary representation of 2 is 10
PS C:\Users\hasin\OneDrive\Desktop\java programming>
```

Both versions correctly convert decimal to binary and handle zero and negative inputs. The zero-shot output (second code) is clearer for beginners because it is written step by step without a function, making the flow easy to follow. The function-based version is more organized and reusable but slightly less direct. Overall, the zero-shot solution is clearer in terms of readability, while the function version is better for reuse.

Question 6: Few-Shot Prompting (Harshad Number Check)

Write a few-shot prompt to generate a Python function that checks whether a number is a Harshad (Niven) number.

Examples:

• Input: 18 → Output: Harshad Number

• Input: 21 → Output: Harshad Number

• Input: 19 → Output: Not a Harshad Number

Task:

• Test boundary conditions.

• Evaluate robustness

```
'''
18
display harshad number
21
display harshad number
19
display not a harshad number

'''
def is_harshad_number(num):
    digit_sum = sum(int(digit) for digit in str(num))
    return num % digit_sum == 0
number = int(input("Enter a number: "))
if is_harshad_number(number):
    print(" harshad number")
else:
    print("not a harshad number")
```

```
gnment 3"
Enter a number: 21
 harshad number
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assi
gnment 3"
Enter a number: 18
 harshad number
PS C:\Users\hasin\OneDrive\Desktop\java programming> & C:\Users\hasin\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/hasin/OneDrive/Desktop/java programming/assi
gnment 3"
Enter a number: 19
not a harshad number
PS C:\Users\hasin\OneDrive\Desktop\java programming>
```

Robustness Evaluation

The program is robust as it safely handles invalid inputs using try-except.
It correctly rejects zero and negative numbers with clear error messages.
Valid Harshad numbers are accurately identified using digit-sum logic.
Edge cases like non-numeric input are handled.