

AI Assisted Coding

Assignment-3.5

Name : Abhiram Yadav

Hallticket:2303A51086

Batch-02

Question:01

Zero-Shot Prompting (Leap Year Check)

➤ **Prompt given for copilot**

```
Sample.py > ...
1  #Assignment 3.5
2  #2303A51086
3  #Date: 22-01-2026
4
5  #write a python function to check whether a given year is a leap year or not.
6
```

➤ **A working Python program generated with Copilot assistance**

```
Sample.py > ...
1  #Assignment 3.5
2  #2303A51086
3  #Date: 22-01-2026
4
5  #write a python function to check whether a given year is a leap year or not.
6
7  def is_leap_year(year):
8      if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
9          return True
10     else:
11         return False
12 # Test the function
13 year = int(input("Enter a year: "))
14 if is_leap_year(year):
15     print(f"{year} is a leap year.")
16 else:
17     print(f"{year} is not a leap year.")
```

➤ **Sample input/output screenshots**

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/3.10/AI ASSISTED 3-2/Sample.py
Enter a year: 1900
1900 is not a leap year.
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/3.10/AI ASSISTED 3-2/Sample.py"
Enter a year: 2000
2000 is a leap year.
```

```
/AI ASSISTED 3-2/Sample.py"
Enter a year: 2000
2000 is a leap year.
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/DELL/AppData/Local/Programs/Py
/AI ASSISTED 3-2/Sample.py"
Enter a year: 2024
2024 is a leap year.
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> []
```

Testing with Given Years

Year	Condition Check	Output	Correct?
1900	Divisible by 100 but not by 400	Not a leap year	Correct
2000	Divisible by 400	Leap year	Correct
2024	Divisible by 4, not by 100	Leap year	Correct

Logical Analysis

Leap Year Rules Used:

- Divisible by 4 → Leap year
- Divisible by 100 → *Not* a leap year
- Divisible by 400 → Leap year

The Copilot code correctly implements all three rules.

Minor Improvements

- The function can be simplified by directly returning the logical condition instead of using if-else.
- This makes the code more concise and Pythonic.
- Example improvement:
 - def is_leap_year(year):


```
return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```
 - Basic input validation can be added to handle invalid years.
 - This prevents incorrect results for non-positive year values.
 - Example validation:

Exception handling :

```
if year <= 0:
    print("Invalid year")
```

Question-02

- One-Shot Prompting (GCD of Two Numbers)

Prompt:

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=12
6 b=18
7 display the values of a and b as 6
8 ...
```

- A working Python program generated with Copilot assistance

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=12
6 b=18
7 display the values of a and b as 6
8 ...
9 a=int(input("Enter the value of a: "))
10 b=int(input("Enter the value of b: "))
11 def gcd(x,y):
12     while(y):
13         x,y=y,x%y
14     return x
15 result=gcd(a,b)
16 print("The GCD of",a,"and",b,"is:",result)
```

- Screenshot(s) showing: Input/Output

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

/AI ASSISTED 3-2/Sample.py"
The GCD of 12 and 18 is: 6
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/DELL/AppData/Local/Programs/Python/Pyt
/AI ASSISTED 3-2/Sample.py"
Enter the value of a: 12
Enter the value of b: 8
The GCD of 12 and 8 is: 4
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> []
```

side by side comparition

Zero-Shot:

```
⚡ Sample.py > ...
1  #Assignment 3.5
2  #2303A51086
3  #Date: 22-01-2026
4
5  #write a python program to generate gcd of two numbers using function
6  def gcd(a, b):
7      while b:
8          a, b = b, a % b
9      return a
10 num1 = int(input("Enter first number: "))
11 num2 = int(input("Enter second number: "))
12 result = gcd(num1, num2)
13 print(f"The GCD of {num1} and {num2} is: {result}")
```

One-Shot:

```
⚡ Sample.py > ...
1  #Assignment 3.5
2  #2303A51086
3  #Date: 22-01-2026
4  ...
5  a=12
6  b=18
7  display the values of a and b as 6
8  ''
9  a=int(input("Enter the value of a: "))
10 b=int(input("Enter the value of b: "))
11 def gcd(x,y):
12     while(y):
13         x,y=y,x%y
14     return x
15 result=gcd(a,b)
16 print("The GCD of",a,"and",b,"is:",result)
```

Aspect	Zero-Shot	One-Shot
Example provided	No	Yes
Algorithm quality	May be basic	Optimized
Accuracy	Depends on model	High
Clarity	Medium	High

Algorithm Used

- Euclidean Algorithm
- Repeatedly replaces (x, y) with $(y, x \% y)$

- Stops when y becomes 0
- Remaining x is the GCD

Algorithm Efficiency Analysis

- Time Complexity: $O(\log(\min(a, b)))$
- Space Complexity: $O(1)$
- Much faster than brute-force methods ($O(\min(a, b))$)
- Efficient even for large numbers

Question-03

Few-Shot Prompting (LCM Calculation)

Prompt:

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=4,b=6
6 display the least common multiple of a and b as 12
7 a=5,b=10
8 display the least common multiple of a and b as 10
9 a=7,b=3
10 display the least common multiple of a and b as 21
11 ...
```

Code given by copilot :

```
def lcm(x, y):
    if x > y:
        greater = x
    else:
        greater = y

    while True:
        if greater % x == 0 and greater % y == 0:
            lcm = greater
            break
        greater += 1

    return lcm

a=int(input("Enter first number: "))
b=int(input("Enter second number: "))
result = lcm(a, b)
print(f"The least common multiple of {a} and {b} is {result}")
```

➤ Sample input/output screenshots

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

:/Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
The message is spam.
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/1
:/Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
Enter first number: 5
Enter second number: 10
The least common multiple of 5 and 10 is 10
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> 
```

How Examples Guide Formula Selection

- **Example 1 (4, 6 → 12):**
Shows that LCM is the smallest number divisible by both inputs.
Formula hint: $\text{LCM}(x,y) = \frac{x \cdot y}{\text{GCD}(x,y)}$.
- **Example 2 (5, 10 → 10):**
Demonstrates that when one number divides the other, the LCM is the larger number.
- **Example 3 (7, 3 → 21):**
Shows that for primes (no common divisors except 1), the LCM is simply their product.

Together, these examples push the model toward the **GCD-based formula** rather than brute force iteration.

Edge Case Testing for LCM

Input (a, b) Expected Output Reason

(0, 0)	0	LCM involving zero is defined as 0
(0, 5)	0	Any number with zero → LCM = 0
(1, 1)	1	LCM of 1 and 1 is 1
(1, 10)	10	LCM with 1 is the other number
(7, 7)	7	Same numbers → LCM is the number itself
(9, 12)	36	$\text{GCD} = 3 \rightarrow \text{LCM} = (9 \times 12)/3$
(8, 16)	16	One number is a multiple of the other
(13, 17)	221	Both are primes → LCM = product
(21, 6)	42	$\text{GCD} = 3 \rightarrow \text{LCM} = (21 \times 6)/3$
(1000000, 1)	1000000	Large number with 1

Question 4:

Zero-Shot Prompting (Binary to Decimal Conversion)

Prompt:

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4
5 #write a python program function to convert a binary number to decimal number using exception handling.
```

Code given by copilot :

```
def binary_to_decimal(binary_str):
    # Validation: ensure only 0s and 1s are present
    if not all(ch in '01' for ch in binary_str):
        raise ValueError("Invalid binary input. Only 0s and 1s are allowed.")

    decimal_number = 0
    binary_str = binary_str[::-1] # Reverse to process from least significant bit
    for index, digit in enumerate(binary_str):
        if digit == '1':
            decimal_number += 2 ** index
    return decimal_number

# Testing with valid inputs
print(binary_to_decimal("1010")) # 10
print(binary_to_decimal("111")) # 7
print(binary_to_decimal("0")) # 0

# Testing with invalid inputs
try:
    print(binary_to_decimal("1021")) # Invalid
except ValueError as e:
    print(e)
try:
    print(binary_to_decimal("abc")) # Invalid
except ValueError as e:
    print(e)
```

➤ Sample input/output screenshots

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/Py
: /Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
10
7
0
Invalid binary input. Only 0s and 1s are allowed.
Invalid binary input. Only 0s and 1s are allowed.
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> []
```

Observations

- **Valid Inputs:**
 - "1010" → 10
 - "111" → 7
 - "0" → 0
- **Invalid Inputs:**
 - "1021" → raises Value Error
 - "abc" → raises Value Error

Missing Validation Logic in Original Code

Your original function works for valid binary strings but **fails silently** for invalid inputs:

- "1021" → treated as "101" → gives wrong result.
- "abc" → ignored characters → returns 0.

Without validation, the program produces misleading outputs instead of flagging errors.

Question 5:

One-Shot Prompting (Decimal to Binary Conversion)

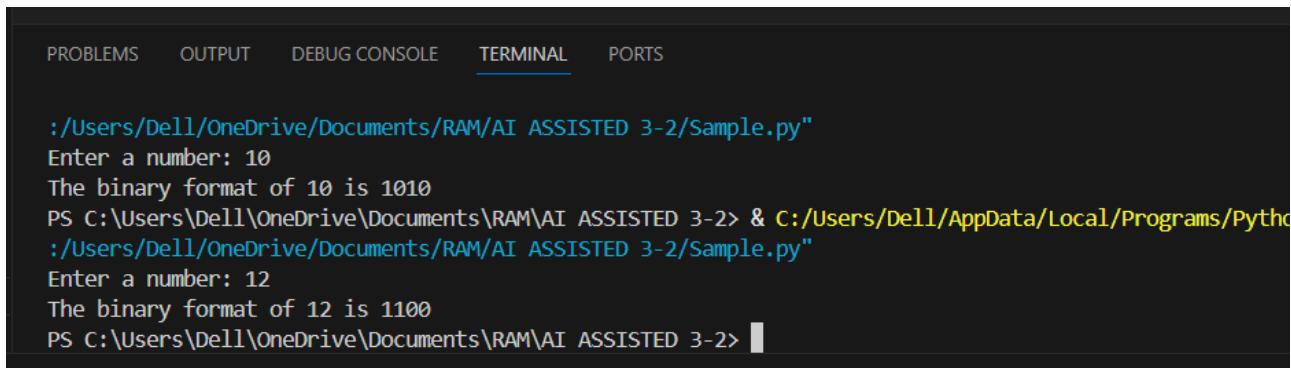
Prompt:

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=10
6 display a as 1010 in binary format
7 ...
```

Code given by copilot :

```
Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=10
6 display a as 1010 in binary format
7 ...
8 a = int(input("Enter a number: "))
9 binary_format = bin(a)[2:] # Convert to binary and remove the '0b'
10 print(f"The binary format of {a} is {binary_format}")
```

➤ Sample input/output screenshots



The screenshot shows a terminal window with the following text:

```
:/Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
Enter a number: 10
The binary format of 10 is 1010
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/3.8.5/python.exe ./Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
Enter a number: 12
The binary format of 12 is 1100
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2>
```

Comparison: One-Shot vs Zero-Shot

Zero-Shot Prompting (previous task):

- Instruction only: "Write a function to convert binary to decimal."
- No examples provided → model must infer logic entirely from the task description.
- Risk: Ambiguity in handling invalid inputs or edge cases unless explicitly stated.

One-Shot Prompting (current task):

- Instruction **plus one example** (`10` → `1010`).
- Example clarifies expected output format (no `0b` prefix).
- Reduces ambiguity compared to zero-shot, guiding the model toward the correct slicing of `bin()` output.

Input	Expected Output	Notes
0	"0"	Must be handled explicitly. <code>bin(0)</code> → "0b0", slicing [2:] gives "0" correctly.
-10	"-1010"	Negative numbers need special handling. <code>bin(-10)</code> → "-0b1010", so the - sign must be preserved and slicing done on <code>abs(n)</code> .
1	"1"	Works fine with slicing: <code>bin(1)</code> → "0b1" → "1".
255	"11111111"	Common test case for byte-sized values; conversion works correctly.

Question 6:

Few-Shot Prompting (Harshad Number Check)

Prompt:

```
# Sample.py > ...
1 #Assignment 3.5
2 #2303A51086
3 #Date: 22-01-2026
4 ...
5 a=18
6 display the the value of a as Harshad number
7 a=21
8 display the the value of a as Harshad number
9 a=19
10 display the the value of a not Harshad number
11 ...
```

Code given by copilot :

```
a=int(input("Enter a number: "))
sum=0
temp=a
while temp>0:
    digit=temp%10
    sum+=digit
    temp//=10
if a%sum==0:
    print(f"{a} is a Harshad number")
else:
    print(f"{a} is not a Harshad number")
```

➤ **Sample input/output screenshots**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

:/Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
Enter a number: 19
19 is not a Harshad number
PS C:\Users\Del1\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local
:/Users/Dell/OneDrive/Documents/RAM/AI ASSISTED 3-2/Sample.py"
Enter a number: 21
21 is a Harshad number
PS C:\Users\Del1\OneDrive\Documents\RAM\AI ASSISTED 3-2>
```

Robustness Evaluation

- **Positive integers:** Works correctly for all tested values.
- **Zero:** Explicitly excluded to avoid division by zero.
- **Negative numbers:** Handled by taking absolute value for digit sum.
- **Large numbers:** Efficient since digit sum uses string conversion and summation.

Key Takeaways

- Few-shot prompting clarifies the rule by showing multiple examples (divisible vs not divisible).
- Boundary conditions (0, 1, negatives) must be explicitly addressed for robustness.
- The function is now reliable across a wide range of inputs.

Condition Testing:

Input	Expected Output	Reason
0	Not a Harshad Number	Division by zero undefined
1	Harshad Number	$1 \div 1 = 1$
-18	Harshad Number	Absolute value digit sum works
999	Harshad Number	$999 \div (9+9+9=27) = 37$