

AI-Assisted Coding

Name: P. Archith

Roll Number: 2303A52258

Batch: 45

Assignment Number: 4.1

Task:1

Prompt: Generate a Python function that checks whether a given year is a leap year with dynamic input.

Code:

```
def is_leap_year(year: int) -> bool:  
    """Return True if year is a leap year (Gregorian rules)."""  
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)  
  
if __name__ == "__main__":  
    while True:  
        s = input("Enter a year (or press Enter to quit): ").strip()  
        if s == "":  
            break  
        try:  
            y = int(s)  
        except ValueError:  
            print("Please enter a valid integer year.")  
            continue  
        if is_leap_year(y):  
            print(f"{y} is a leap year.")  
        else:  
            print(f"{y} is not a leap year.")
```

Output:

```
Z\AI Assitant Coding\Task3-5.py
Enter a year (or press Enter to quit): 2000
2000 is a leap year.
Enter a year (or press Enter to quit): 2024
2024 is a leap year.
Enter a year (or press Enter to quit): 2025
2025 is not a leap year.
Enter a year (or press Enter to quit):
○ (.venv) PS D:\3-2\AI Assitant Coding> 
```

Task 2:

Prompt: Write a Python code gcd(a, b) that returns the greatest common divisor of two positive integers using an efficient algorithm.

Example:

Input: a = 12, b = 18

Output: 6

Now generate the full code.

Code:

```
def gcd(a: int, b: int) -> int:
    """Return the greatest common divisor of integers a and b using the Euclidean algorithm."""
    if not isinstance(a, int) or not isinstance(b, int):
        raise TypeError("a and b must be integers")
    a, b = abs(a), abs(b)
    if a == 0:
        return b
    if b == 0:
        return a
    while b:
        a, b = b, a % b
    return a

if __name__ == "__main__":
    # Example usage:
    print("GCD: ", gcd(12, 18)) # Output: 6
```

Output:

```
Z\AI Assitant Coding\Task3-5.py
GCD:  6
○ (.venv) PS D:\3-2\AI Assitant Coding> 
```

Task 3:

Prompt: Write a Python code to find $\text{lcm}(a, b)$ that returns the Least Common Multiple (LCM) of two positive integers.

Example 1:

Input: 4, 6

Output: 12

Example 2:

Input: 5, 10

Output: 10

Example 3:

Input: 7, 3

Output: 21

Now generate the Python code.

Code:

```
def gcd(a: int, b: int) -> int:
    if not isinstance(a, int) or not isinstance(b, int):
        raise TypeError("a and b must be integers")
    a, b = abs(a), abs(b)
    while b:
        a, b = b, a % b
    return a

def lcm(a: int, b: int) -> int:
    if not isinstance(a, int) or not isinstance(b, int):
        raise TypeError("a and b must be integers")
    if a == 0 or b == 0:
        return 0
    return abs(a // gcd(a, b) * b)

if __name__ == "__main__":
    print("Lcm of 4,6: ",lcm(4, 6)) # 12
    print("Lcm of 5,10: ",lcm(5, 10)) # 10
    print("Lcm of 7,3: ",lcm(7, 3)) # 21
```

Output:

```
ask3-5.py'
Lcm of 4,6: 12
Lcm of 5,10: 10
Lcm of 7,3: 21
○ (.venv) PS D:\3-2\AI Assitant Coding> 
```

Task 4:

Prompt: Write a Python code to find binary_to_decimal(binary_str) that converts a binary number (given as a string) into its decimal equivalent and returns the result.

Code:

```
def binary_to_decimal(binary_str: str) -> int:
    """Convert a binary string to its decimal integer equivalent.

    Accepts optional leading sign (+/-) and optional 0b/0B prefix.

    Raises TypeError if input is not a str and ValueError for invalid binary.

    """
    if not isinstance(binary_str, str):
        raise TypeError("binary_str must be a string")
    s = binary_str.strip()
    if not s:
        raise ValueError("empty binary string")
    sign = 1
    if s[0] in "+-":
        sign = -1 if s[0] == "-" else 1
    s = s[1:]
    if not s:
        raise ValueError("invalid binary string")
    if s.startswith(("0b", "0B")):
        s = s[2:]
    if not s:
        raise ValueError("empty binary string")
```

```
if any(c not in "01" for c in s):  
    raise ValueError("invalid binary string")  
  
result = 0  
  
for ch in s:  
    result = (result << 1) + int(ch)  
  
return sign * result
```

```
if __name__ == "__main__":  
    print(binary_to_decimal("1011")) # 11  
    print(binary_to_decimal("-0b1011")) # -11  
    print(binary_to_decimal("0")) # 0
```

output:

```
ask3-5.py'  
11  
-11  
0  
○ (.venv) PS D:\3-2\AI Assitant Coding> █
```

Explanations:

Missing Validation Logic (Typical in Zero-Shot Outputs)

Because the prompt gives **no examples or constraints**, the generated function may:

- Assume all inputs are valid binary strings
- Skip checks for characters other than 0 and 1
- Fail silently or raise unhandled exceptions on invalid input
- Not handle empty strings or whitespace properly

Common validation that may be missing:

- Verifying input type is str
- Ensuring all characters $\in \{0,1\}$
- Handling empty input
- Providing clear error messages (e.g., ValueError)

Task-5:

Prompt: Write a Python code to find decimal_to_binary(n) that converts a non-negative integer into its binary representation and returns it as a string.

Example:

Input: 10

Output: "1010"

Now generate the code.

Code:

```
def decimal_to_binary(n: int) -> str:
    """Convert a non-negative integer to its binary representation as a string."""
    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n < 0:
        raise ValueError("n must be non-negative")
    if n == 0:
        return "0"
    bits = []
    while n:
        bits.append(str(n & 1))
        n >>= 1
    return ''.join(reversed(bits))

if __name__ == "__main__":
    print(decimal_to_binary(10)) # 1010
```

Output:

```
ask3-5.py'
1010
(.venv) PS D:\3-2\AI Assistant Coding>
```

Explanations:

Comparison with Zero-Shot Output (Clarity)

- The **example removes ambiguity** about the output format (string vs integer).
- It clearly signals **base-2 conversion logic**, avoiding incorrect approaches (e.g., returning lists or prefixes like 0b).
- One-shot prompting typically yields **more consistent and correct implementations** than zero-shot prompting.

Handling of Zero and Negative Numbers

- **Zero (n = 0)**
 - Should explicitly return "0"
 - Without clarification, some models may return an empty string.
- **Negative Numbers (n < 0)**

- Not defined in the prompt, so behavior may vary:
 - Some implementations may fail
 - Others may ignore the sign
- A robust function should either:
 - Raise a ValueError, or
 - Handle the sign separately (e.g., -10 → "-1010")

Task-6:

Prompt: Write a Python code to find `is_harshad(n)` that checks whether a given positive integer is a Harshad (Niven) number.

A Harshad number is a number that is divisible by the sum of its digits.

Example 1:

Input: 18

Output: Harshad Number

Example 2:

Input: 21

Output: Harshad Number

Example 3:

Input: 19

Output: Not a Harshad Number

Now generate the Python code.

Code:

```
def is_harshad(n: int) -> bool:
    """Return True if n is a Harshad (Niven) number: divisible by the sum of its digits."""
    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n <= 0:
        raise ValueError("n must be a positive integer")
    digit_sum = sum(int(d) for d in str(abs(n)))
    if digit_sum == 0:
        return False
    return n % digit_sum == 0

if __name__ == "__main__":
    for val in (18, 21, 19):
        print(f"{val}: {'Harshad Number' if is_harshad(val) else 'Not a Harshad Number'}")
```

Output:

```
ask3-5.py'
18: Harshad Number
21: Harshad Number
19: Not a Harshad Number
○ (.venv) PS D:\3-2\AI Assistant Coding> 
```

Explanations:

Boundary Conditions to Test

- $n = 1 \rightarrow$ Harshad Number ($1 \div 1 = 1$)
 - $n = 0 \rightarrow$ Undefined (division by zero risk)
 - Single-digit numbers (2–9) \rightarrow All Harshad Numbers
 - Very large numbers (e.g., 10^{12})
 - Negative numbers (-18) \rightarrow Not defined unless specified
-

Robustness Evaluation

- The examples clearly guide the model to:
 - Compute digit sum correctly
 - Use modulus ($n \% \text{digit_sum} == 0$)
- **Potential weaknesses without extra constraints:**
 - Missing validation for $n \leq 0$
 - No handling of non-integer input
 - Output formatting may vary if not enforced strictly

To improve robustness, the prompt could add:

- Input constraints ($n > 0$)
- Explicit error handling or boolean return type