Lab Assignment 3.5

Name:M.Hasini

HT.no:2303A51109

Batch:02

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

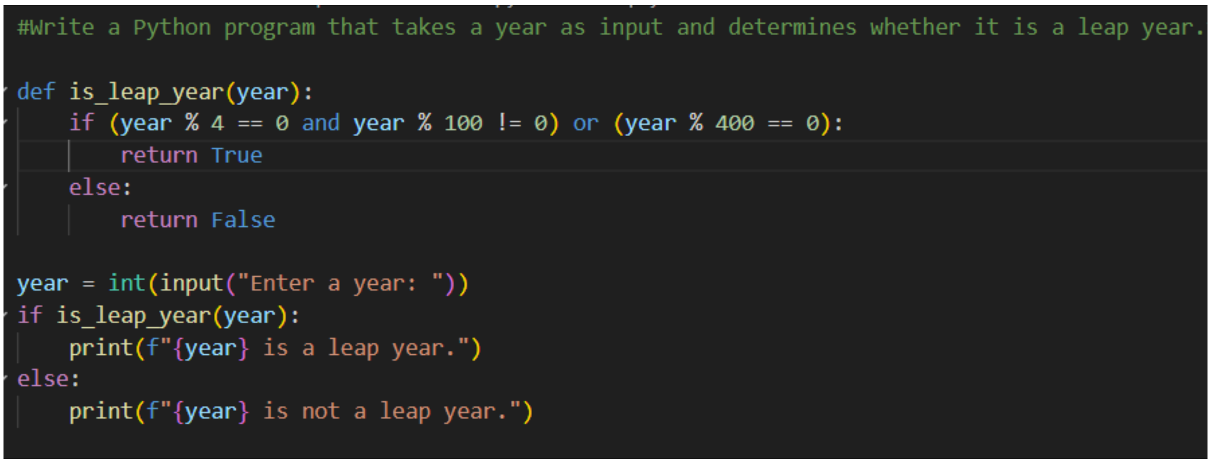
Write a zero-shot prompt to generate a Python function that checks whether a given year is a leap year.

Task:

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

Prompt:

#write a python function that checks wheather a given year is leap year

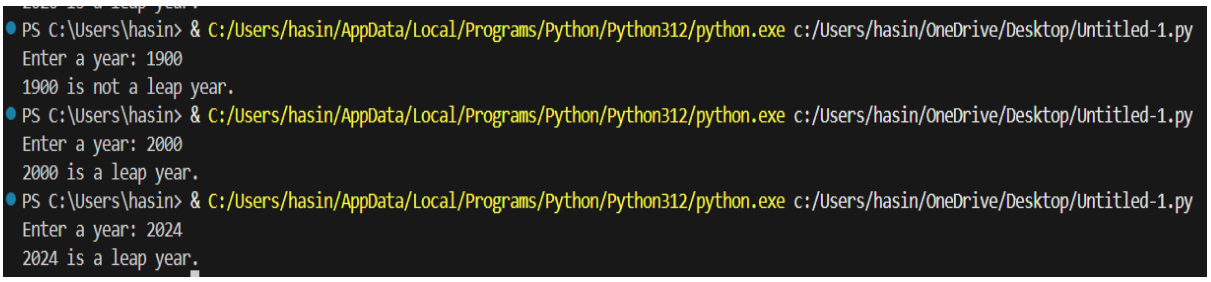


```
#Write a Python program that takes a year as input and determines whether it is a leap year.

def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Code:

```
#write a python function that checks wheather a given year is leap year ..
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
try:
    year = int(input("Enter a year to check if it's a leap year: "))
    if year < 0:
        print("Invalid input. Please enter a positive integer for the year.")
    else:
        if is_leap_year(year):
            print(f"{year} is a leap year.")
        else:
            print(f"{year} is not a leap year.")
except ValueError:
    print("Invalid input. Please enter a valid integer for the year.")
```

input/ouput:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/OneDrive/Desktop/Untitled-1.py
Enter a year: 1900
1900 is not a leap year.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/OneDrive/Desktop/Untitled-1.py
Enter a year: 2000
2000 is a leap year.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/OneDrive/Desktop/Untitled-1.py
Enter a year: 2024
2024 is a leap year.
```

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:

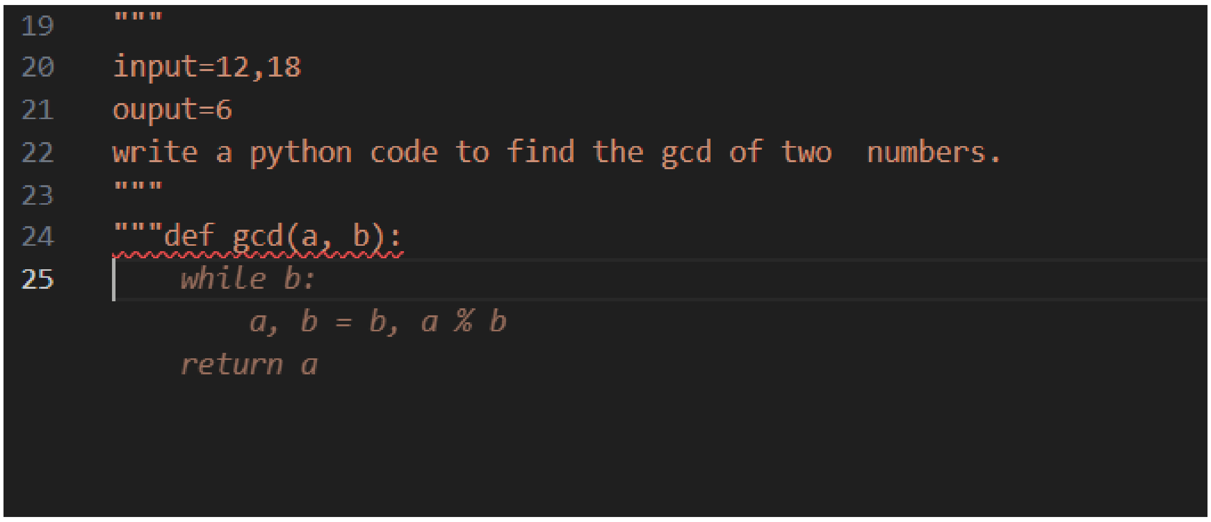
- Compare with a zero-shot solution.
- Analyze algorithm efficiency.

Prompt:

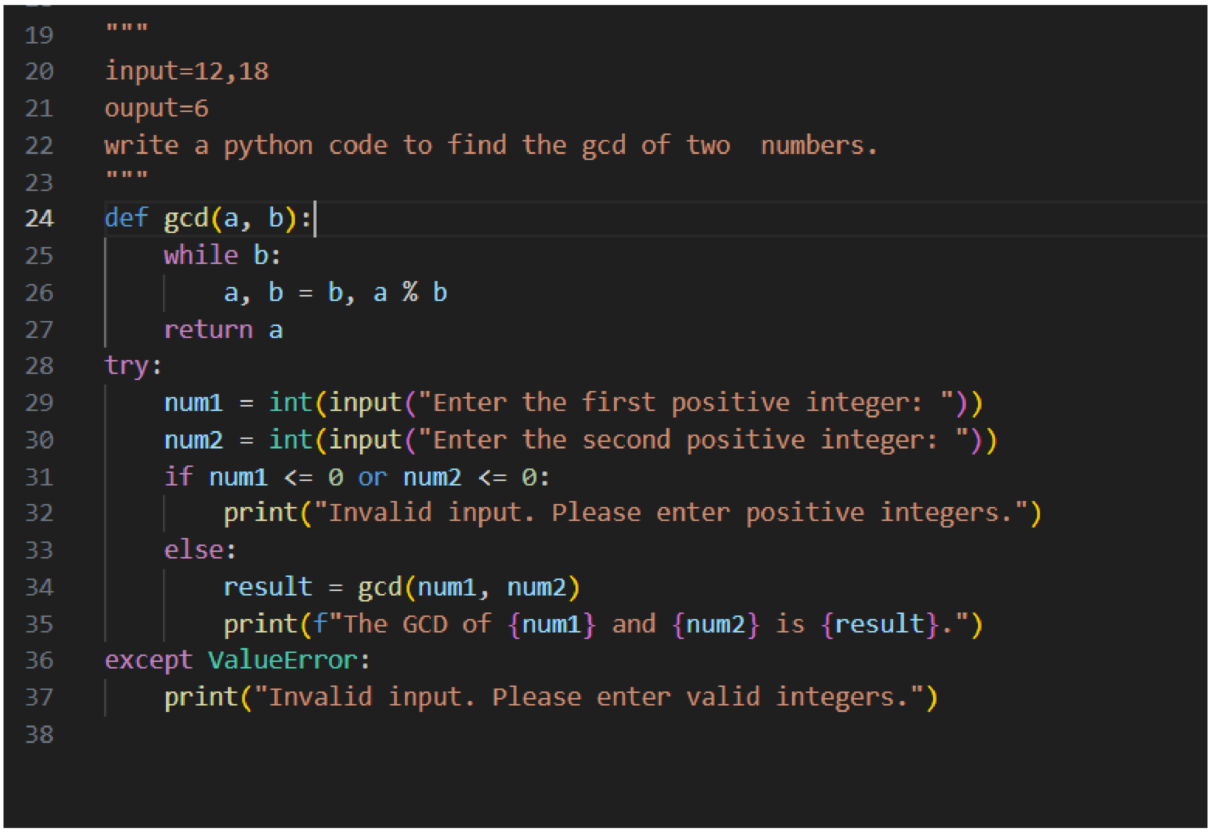
input=12,18

ouput=6

write a python code to find the gcd of two numbers.



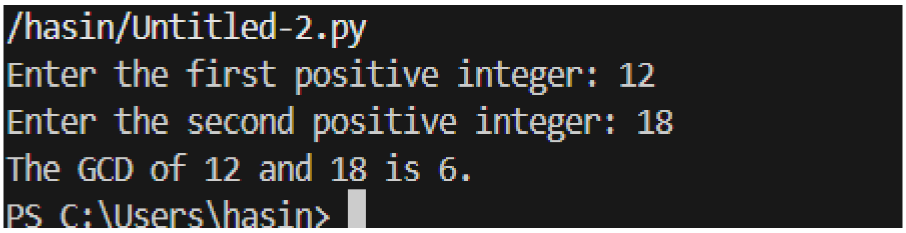
```
"""
input=12,18
ouput=6
write a python code to find the gcd of two  numbers.
"""
"""def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
```



```
"""
input=12,18
ouput=6
write a python code to find the gcd of two  numbers.
"""
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
try:
    num1 = int(input("Enter the first positive integer: "))
    num2 = int(input("Enter the second positive integer: "))
    if num1 <= 0 or num2 <= 0:
        print("Invalid input. Please enter positive integers.")
    else:
        result = gcd(num1, num2)
        print(f"The GCD of {num1} and {num2} is {result}.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

Code:

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
try:
    num1 = int(input("Enter the first positive integer: "))
    num2 = int(input("Enter the second positive integer: "))
    if num1 <= 0 or num2 <= 0:
        print("Invalid input. Please enter positive integers.")
    else:
        result = gcd(num1, num2)
        print(f"The GCD of {num1} and {num2} is {result}.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```



```
/hasin/Untitled-2.py
Enter the first positive integer: 12
Enter the second positive integer: 18
The GCD of 12 and 18 is 6.
PS C:\Users\hasin>
```

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

A zero-shot solution for finding the GCD checks all numbers from 1 to the minimum of the two inputs and finds the greatest common divisor, which takes linear time. In contrast, the given code uses the Euclidean algorithm, which repeatedly reduces the problem using the modulo operation. This approach is much more efficient, with a time complexity of O(log min(a, b)), while both methods use constant space. Hence, the implemented solution is faster and more suitable for large inputs.

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.
- Prompt:

input=4,6

ouput=12

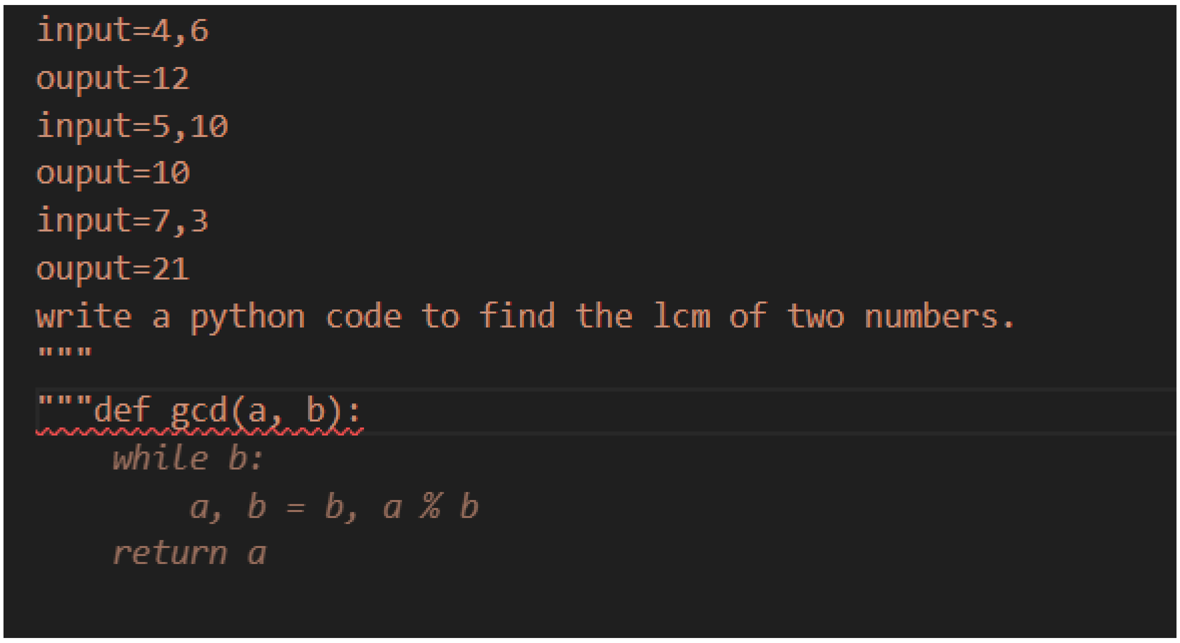
input=5,10

ouput=10

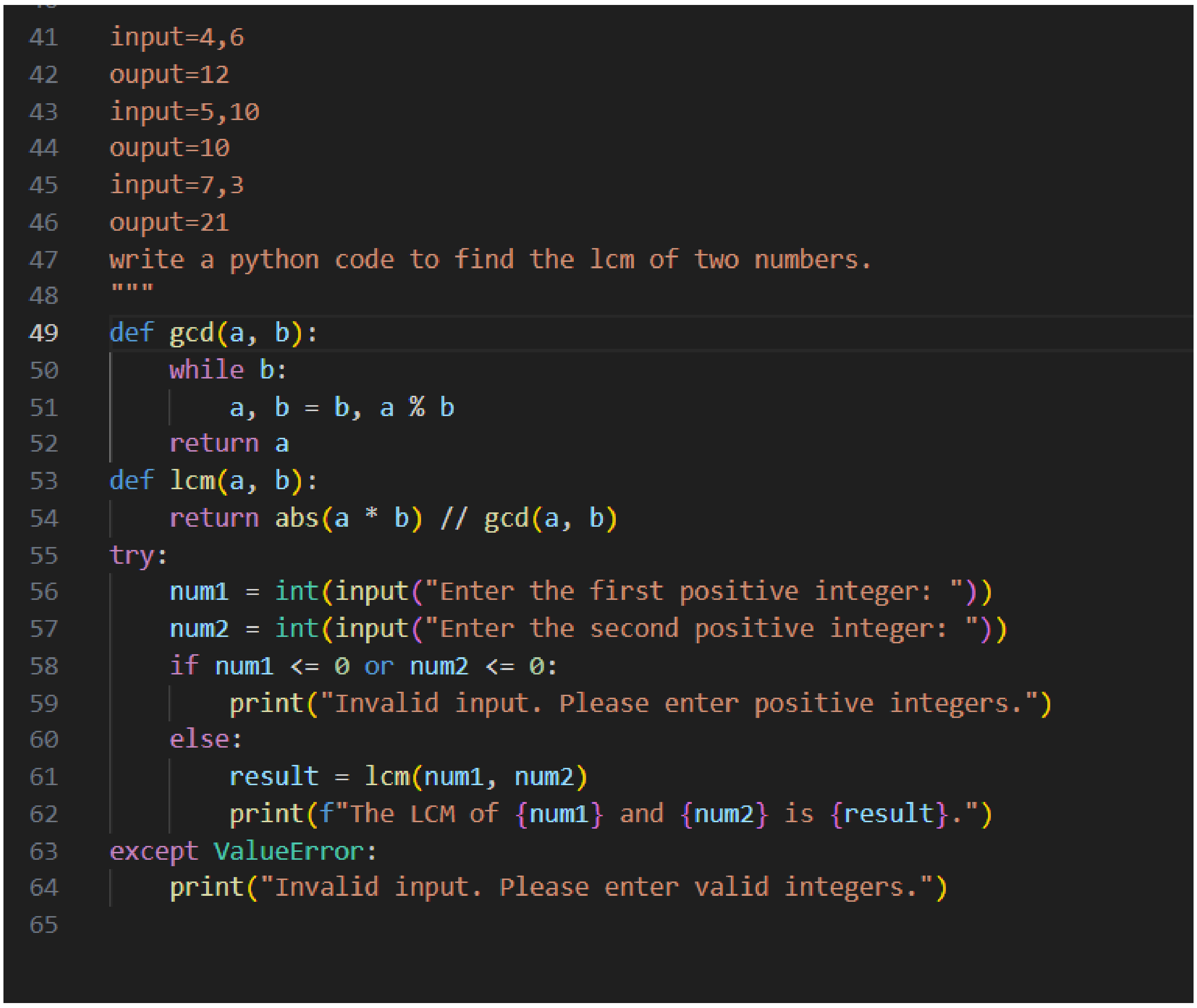
input=7,3

ouput=21

write a python code to find the lcm of two numbers.



```
input=4,6
ouput=12
input=5,10
ouput=10
input=7,3
ouput=21
write a python code to find the lcm of two numbers.
"""
"""def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
```

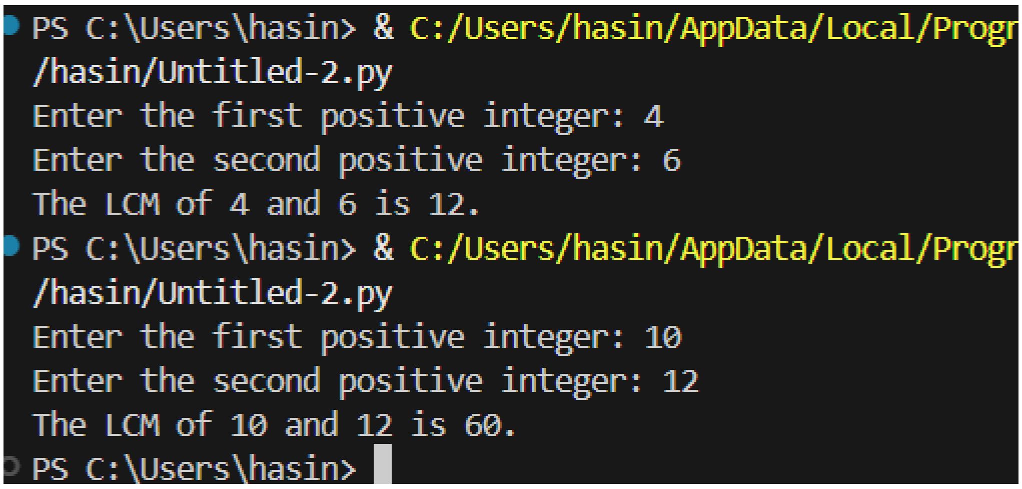


```
input=4,6
ouput=12
input=5,10
ouput=10
input=7,3
ouput=21
write a python code to find the lcm of two numbers.
"""
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def lcm(a, b):
    return abs(a * b) // gcd(a, b)
try:
    num1 = int(input("Enter the first positive integer: "))
    num2 = int(input("Enter the second positive integer: "))
    if num1 <= 0 or num2 <= 0:
        print("Invalid input. Please enter positive integers.")
    else:
        result = lcm(num1, num2)
        print(f"The LCM of {num1} and {num2} is {result}.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

Code:

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def lcm(a, b):
    return abs(a * b) // gcd(a, b)
try:
    num1 = int(input("Enter the first positive integer: "))
    num2 = int(input("Enter the second positive integer: "))
    if num1 <= 0 or num2 <= 0:
        print("Invalid input. Please enter positive integers.")
    else:
        result = lcm(num1, num2)
        print(f"The LCM of {num1} and {num2} is {result}.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

input/output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Progr
/hasin/Untitled-2.py
Enter the first positive integer: 4
Enter the second positive integer: 6
The LCM of 4 and 6 is 12.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Progr
/hasin/Untitled-2.py
Enter the first positive integer: 10
Enter the second positive integer: 12
The LCM of 10 and 12 is 60.
PS C:\Users\hasin>
```

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

From the given examples (4,6 → 12; 5,10 → 10; 7,3 → 21), we can observe that the LCM is the smallest number divisible by both inputs, which guides us to use the standard formula LCM(a, b) = (a × b) / GCD(a, b). The code correctly applies this formula using the Euclidean algorithm for efficiency. Edge cases such as zero, negative values, and non-integer inputs are handled by input validation, ensuring the program works only for positive integers and avoids invalid results.

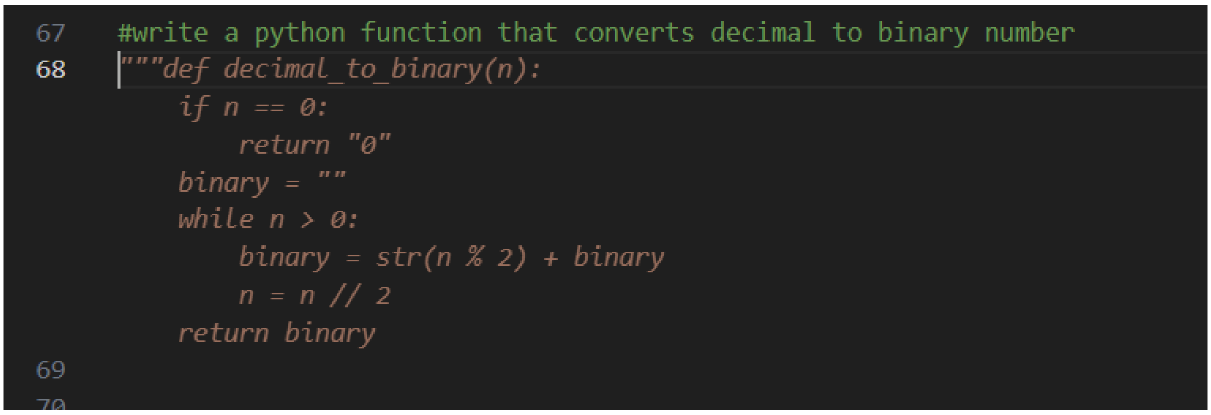
Question 4: Zero-Shot Prompting (Binary to Decimal Conversion)

Write a zero-shot prompt to generate a Python function that converts a binary number to decimal.

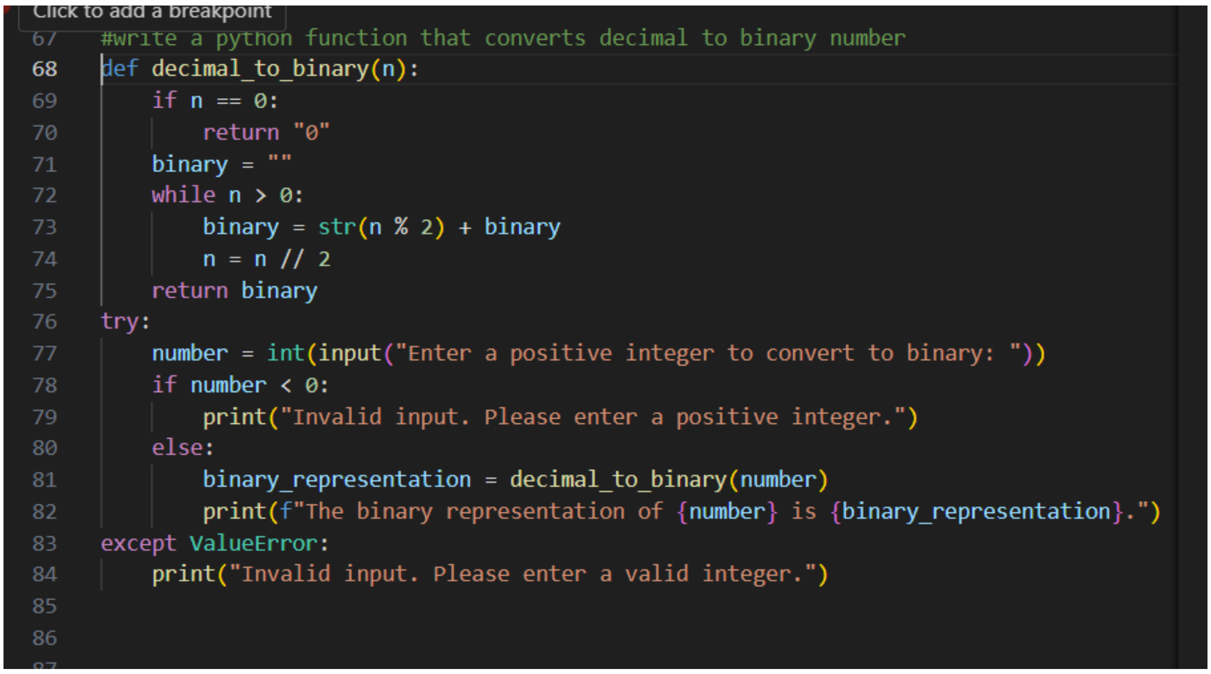
Task:

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

Prompt: #write a python function that converts decimal to binary number



```
#write a python function that converts decimal to binary number
"""def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
```

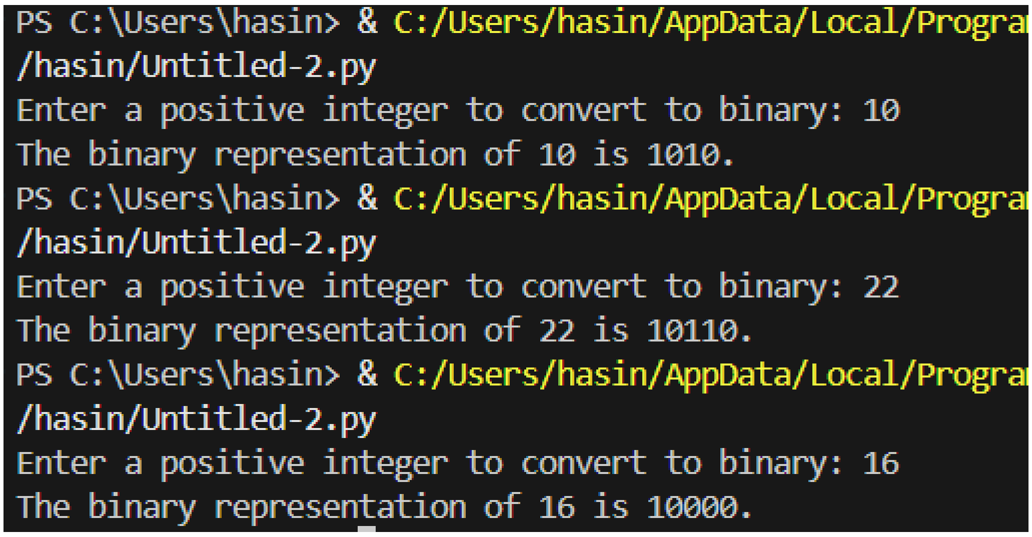


```
#write a python function that converts decimal to binary number
def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
try:
    number = int(input("Enter a positive integer to convert to binary: "))
    if number < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        binary_representation = decimal_to_binary(number)
        print(f"The binary representation of {number} is {binary_representation}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Code:

```
def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
try:
    number = int(input("Enter a positive integer to convert to binary: "))
    if number < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        binary_representation = decimal_to_binary(number)
        print(f"The binary representation of {number} is {binary_representation}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

input/ouput:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Progra
/hasin/Untitled-2.py
Enter a positive integer to convert to binary: 10
The binary representation of 10 is 1010.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Progra
/hasin/Untitled-2.py
Enter a positive integer to convert to binary: 22
The binary representation of 22 is 10110.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Progra
/hasin/Untitled-2.py
Enter a positive integer to convert to binary: 16
The binary representation of 16 is 10000.
```

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.

Example:
Input: 10 → Output: 1010

Task:

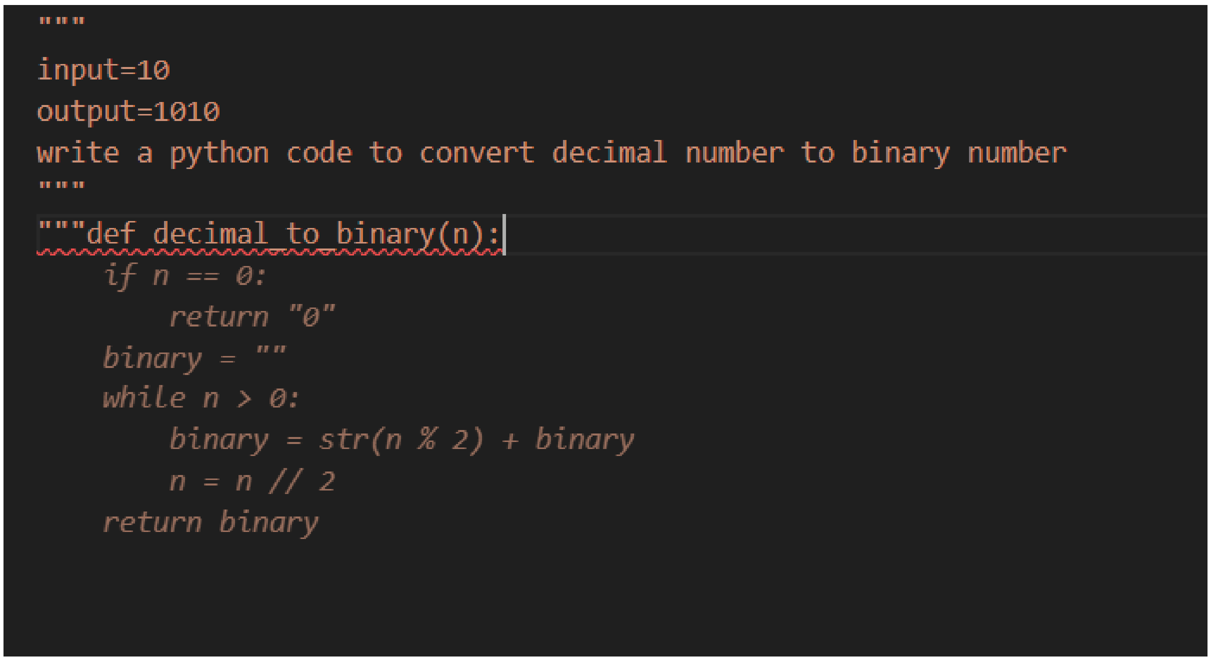
- Compare clarity with zero-shot output.
- Analyze handling of zero and negative numbers.

Prompt:

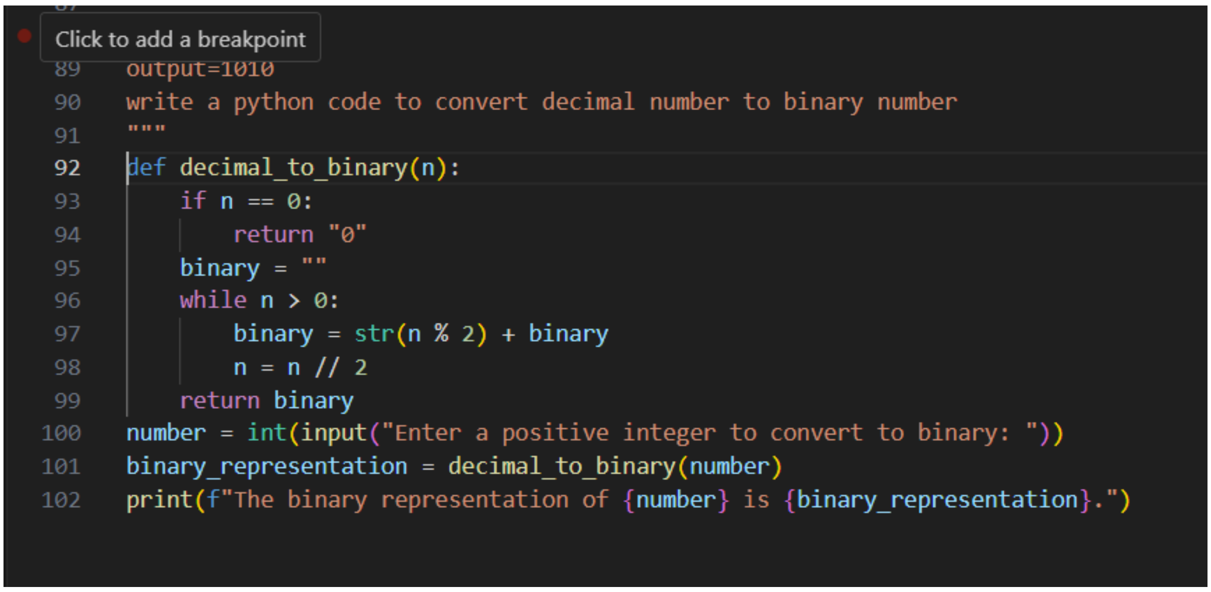
input=10

output=1010

write a python code to convert decimal number to binary number



```
"""
input=10
output=1010
write a python code to convert decimal number to binary number
"""
"""def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
```

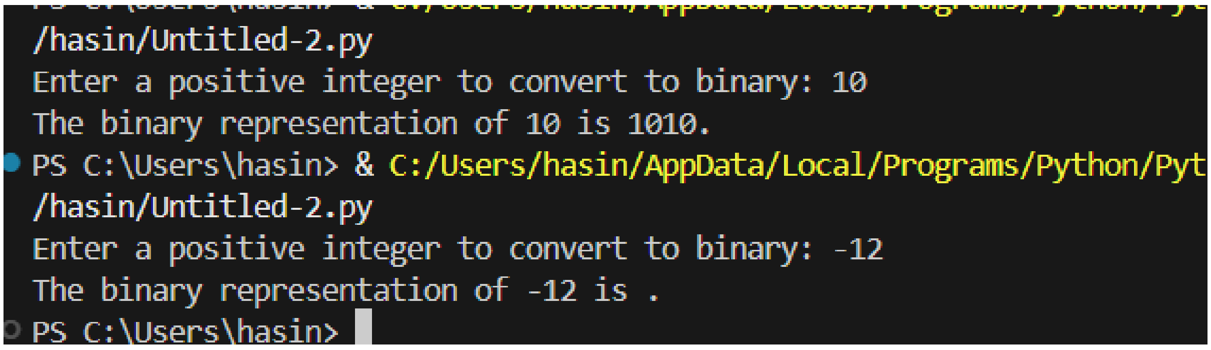


```
output=1010
write a python code to convert decimal number to binary number
"""
def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
number = int(input("Enter a positive integer to convert to binary: "))
binary_representation = decimal_to_binary(number)
print(f"The binary representation of {number} is {binary_representation}.")
```

Code:

```
def decimal_to_binary(n):
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary
number = int(input("Enter a positive integer to convert to binary: "))
binary_representation = decimal_to_binary(number)
print(f"The binary representation of {number} is {binary_representation}.")
```

Inputs/Outputs:



```
/hasin/Untitled-2.py
Enter a positive integer to convert to binary: 10
The binary representation of 10 is 1010.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Pyt
/hasin/Untitled-2.py
Enter a positive integer to convert to binary: -12
The binary representation of -12 is .
PS C:\Users\hasin>
```

Task:

- Test boundary conditions.
- Evaluate robustness

Compared to a zero-shot output, this implementation is clearer and more readable because it explicitly shows each step of the decimal-to-binary conversion process. However, while the function correctly handles the case when the input is zero by returning "0", it does not handle negative numbers. Without validation, negative inputs would result in incorrect behavior, so additional checks are needed to restrict the input to non-negative integers.

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

Prompt:

input=18

ouput=harshad number

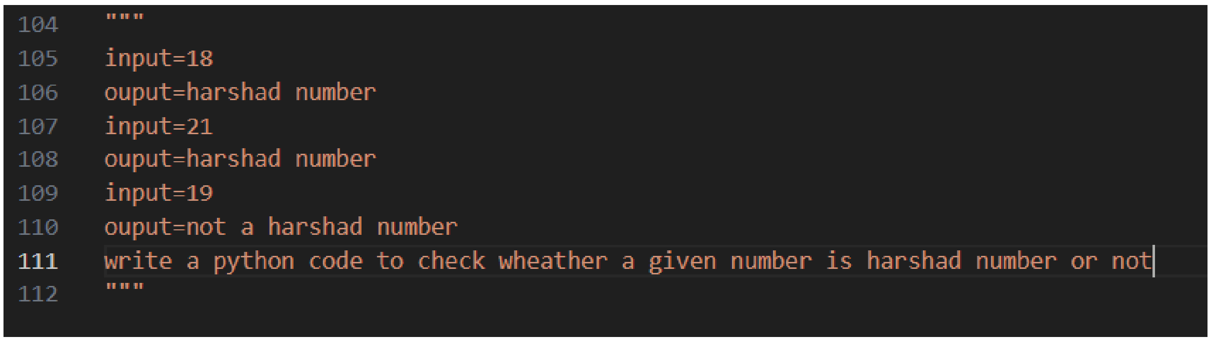
input=21

ouput=harshad number

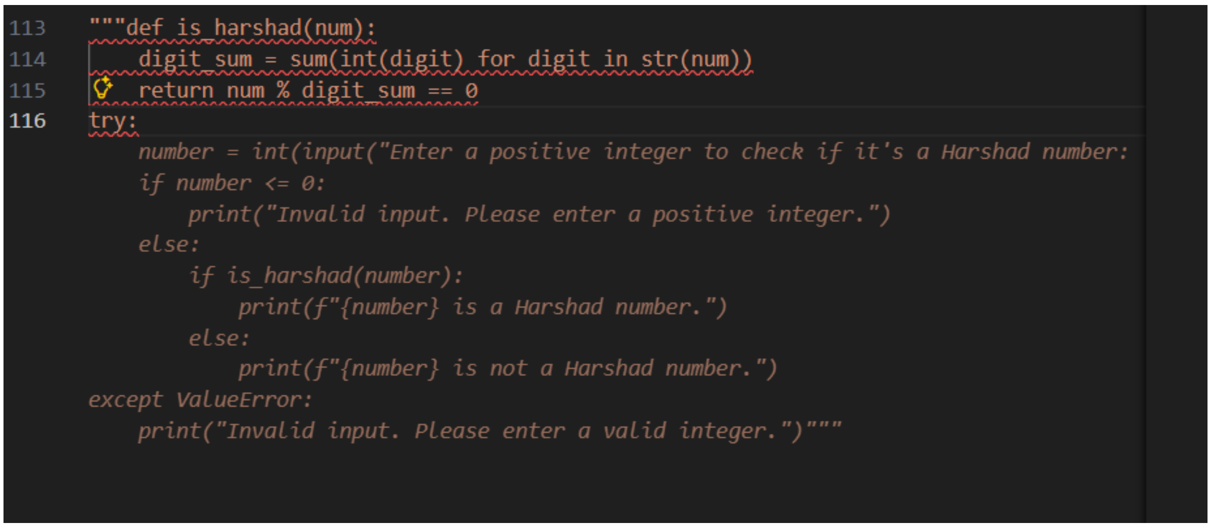
input=19

ouput=not a harshad number

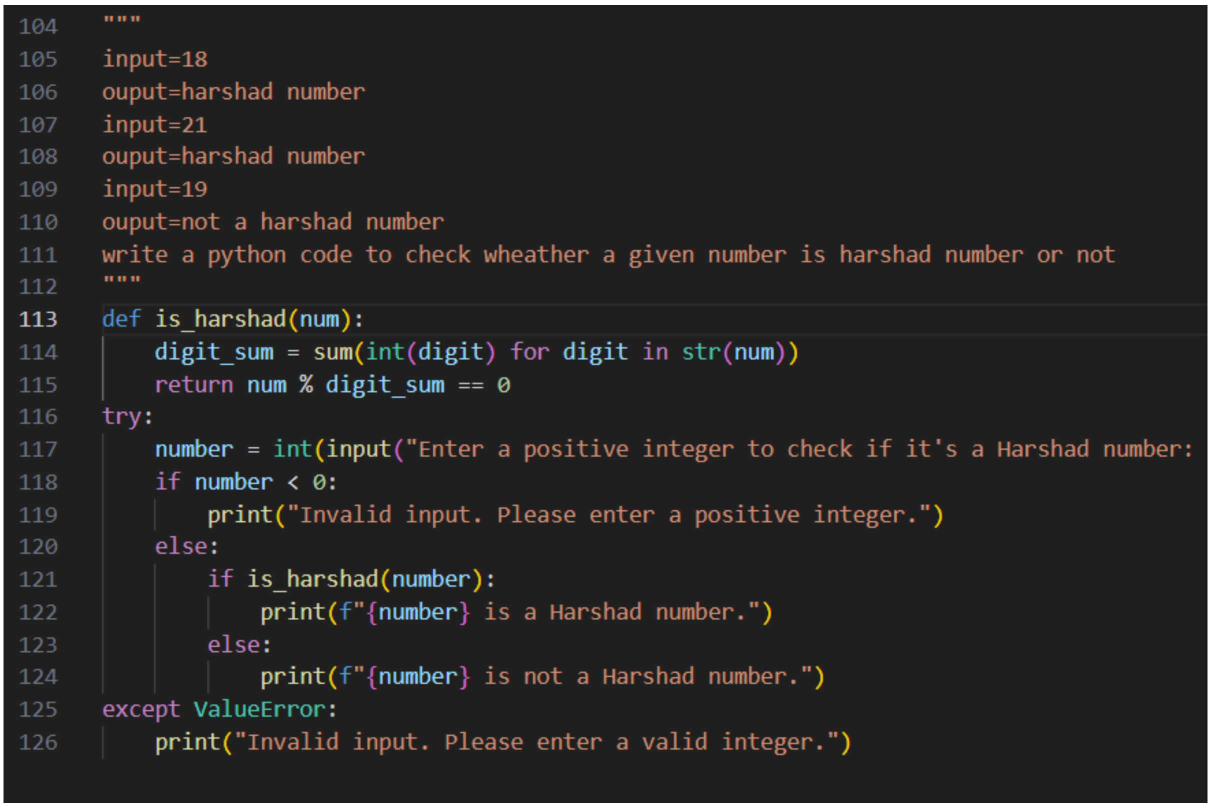
write a python code to check wheather a given number is harshad number or not



```
"""
input=18
ouput=harshad number
input=21
ouput=harshad number
input=19
ouput=not a harshad number
write a python code to check wheather a given number is harshad number or not
"""
```



```
"""def is_harshad(num):
    digit_sum = sum(int(digit) for digit in str(num))
    return num % digit_sum == 0
try:
    number = int(input("Enter a positive integer to check if it's a Harshad number:
    if number <= 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_harshad(number):
            print(f"{number} is a Harshad number.")
        else:
            print(f"{number} is not a Harshad number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")"""
```

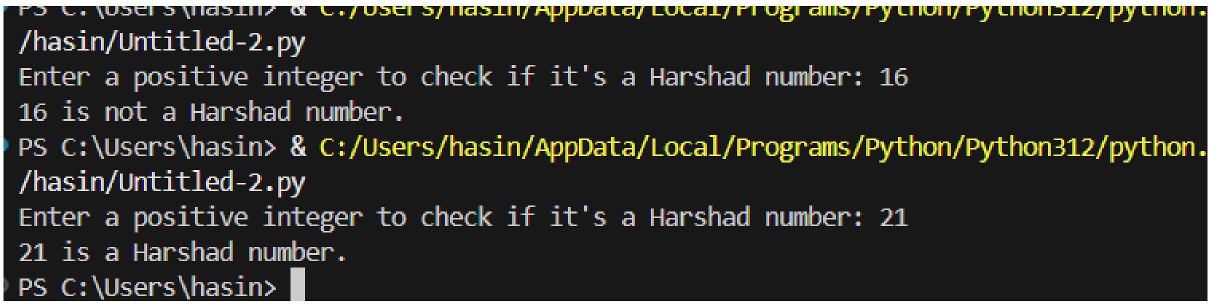


```python
"""
input=18
ouput=harshad number
input=21
ouput=harshad number
input=19
ouput=not a harshad number
write a python code to check wheather a given number is harshad number or not
"""
def is_harshad(num):
    digit_sum = sum(int(digit) for digit in str(num))
    return num % digit_sum == 0
try:
    number = int(input("Enter a positive integer to check if it's a Harshad number:
    if number < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_harshad(number):
            print(f"{number} is a Harshad number.")
        else:
            print(f"{number} is not a Harshad number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Code:

```python
def is_harshad(num):
    digit_sum = sum(int(digit) for digit in str(num))
    return num % digit_sum == 0
try:
    number = int(input("Enter a positive integer to check if it's a Harshad number: "))
    if number < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_harshad(number):
            print(f"{number} is a Harshad number.")
        else:
            print(f"{number} is not a Harshad number.")
except ValueError:
```

input/output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.
/hasin/Untitled-2.py
Enter a positive integer to check if it's a Harshad number: 16
16 is not a Harshad number.
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.
/hasin/Untitled-2.py
Enter a positive integer to check if it's a Harshad number: 21
21 is a Harshad number.
PS C:\Users\hasin>
```

Task:

- Test boundary conditions.
- Evaluate robustness

The code correctly identifies Harshad numbers for valid inputs such as 18 and 21, and correctly reports non-Harshad numbers like 19. It also handles invalid inputs such as negative numbers and non-integer values using input validation and exception handling. However, the logic does not explicitly handle the edge case where the input is 0, which would cause a division by zero error. Adding a specific check to reject 0 would complete the validation logic and make the program more robust.