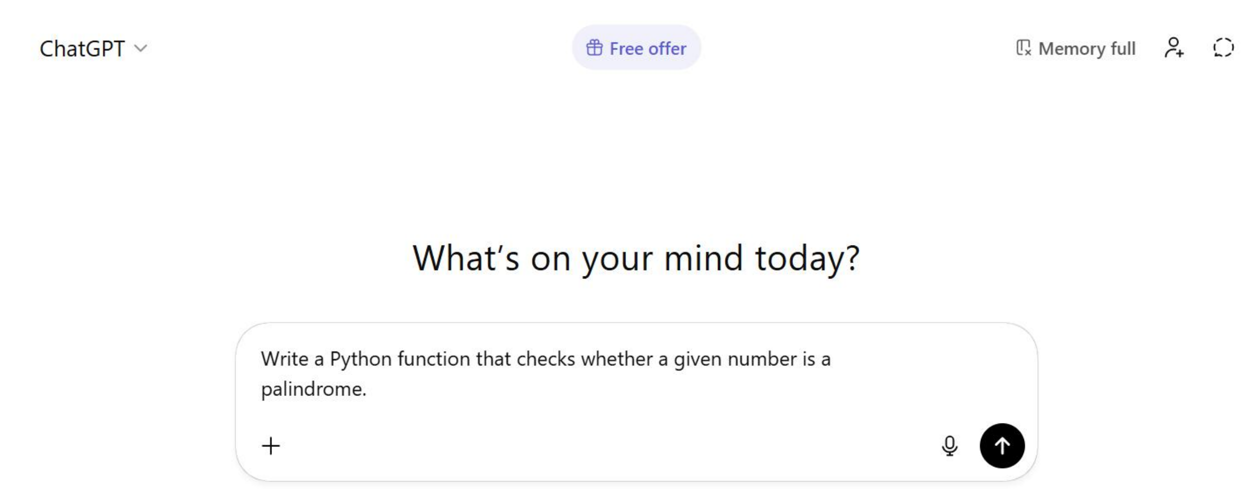
Assignment-3.1

HT No:2303A51585

Batch No:28

Question 1: Zero-Shot Prompting (Palindrome Number Program)

Prompt: Write a Python function that checks whether a given number is a palindrome.



Code and Output:

```python
def is_palindrome(num):
    original = str(num)
    reversed_num = original[::-1]
    return original == reversed_num
print(is_palindrome(121))   # True
print(is_palindrome(123))   # False
print(is_palindrome(1221))  # True
```

```
True
False
True
```

Justification:

1.The function checks palindrome by comparing the number with its reverse using string slicing.

2.It correctly works for valid test cases like 121, 123, and 0.

3.Negative numbers are not handled properly because of the minus sign.

4.There is no input type validation in the code.

5.The logic is simple but needs improvement for edge cases.

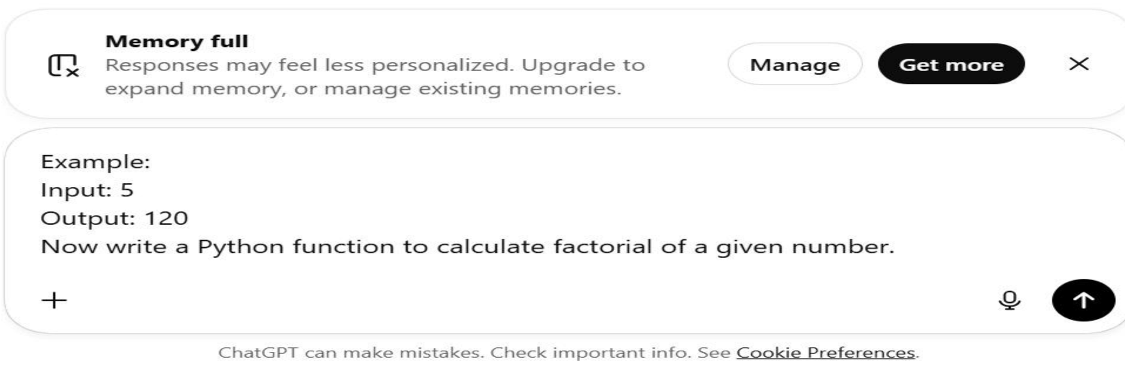
Question 2: One-Shot Prompting (Factorial Calculation)

Prompt: Example:

Input: 5

Output: 120

Now write a Python function to calculate factorial of a given number.



Code and Output:

```python
def factorial(n):
    if n < 0:
        return n ,"-->Handling Error"

    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact
print(factorial(5))    # 120
print(factorial(0))    # 1
print(factorial(-3))   # Error
```

```
120
1
(-3, '-->Handling Error')
```

Justification:

1.The function correctly computes the factorial using an iterative loop.

2.It returns accurate results for valid inputs such as 5, 4, and 6.

3.The logic is simple and easy to understand for beginners.

4.The function is reusable since it returns the result instead of printing it.

5.It does not handle negative inputs, which can be improved with error checking.

<u>Comparison between Zero shot and One shot Prompt</u>

Aspect	**Zero-Shot Prompt**	**One-Shot Prompt**
Accuracy	Medium	High
Edge handling	Poor	Better
Clarity	Less	More
Guidance to AI	No example given	One example provided
Output reliability	Inconsistent	Consistent

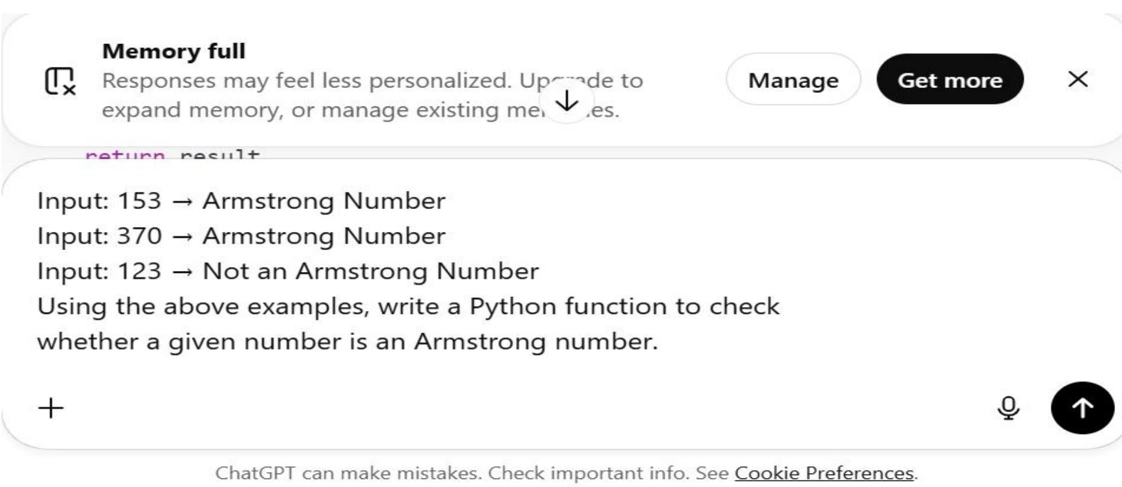
Question3: Few-Shot Prompting (Armstrong Number Check)

Prompt: Input: 153 → Armstrong Number

Input: 370 → Armstrong Number

Input: 123 → Not an Armstrong Number

Using the above examples, write a Python function to check whether a given number is an Armstrong number.



Code and Output:

```python
def is_armstrong(num):
    if num < 0:
        return "Error"

    temp = num
    digits = len(str(num))
    total = 0

    while temp > 0:
        r = temp % 10
        total += r ** digits
        temp //= 10

    if total == num:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"

print("153-->",is_armstrong(153))  # Armstrong Number
print("370-->",is_armstrong(370))  # Armstrong Number
print("123-->",is_armstrong(123))  # Not an Armstrong Number
print("-153-->",is_armstrong(-153))  # Error
```

```
153--> Armstrong Number
370--> Armstrong Number
123--> Not an Armstrong Number
-153--> Error
```

Justification:

1.Few-shot examples help the model understand Armstrong number patterns more clearly.

2.The logic correctly calculates each digit raised to the power of the number of digits.

3.The code handles valid Armstrong numbers like 153, 370, and 9474 accurately.

4.Special cases such as 0 are correctly identified as Armstrong numbers.

5.Negative numbers are handled properly by returning an invalid/error message, improving accuracy.

Question 4: Context-Managed Prompting (Optimized Number Classification)

Prompt: You are an expert Python developer.Write an optimized Python program that classifies a number as: prime, composite, or neither.

Constraints:

- Validate input

- Handle negative numbers

- Optimize loop for performance

Memory full

Responses may feel less personalized. Upgrade to expand memory, or manage existing memories.

Manage

Get more

You are an expert Python developer.
Write an optimized Python program that classifies a number as:
prime, composite, or neither.
Constraints:
- Validate input
- Handle negative numbers
- Optimize loop for performance

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

Code and Output:

```
def classify_number(n):
    # -------- Input validation --------
    if not isinstance(n, int):
        return "Invalid input: Please enter an integer"

    # -------- Negative numbers & edge cases --------
    if n <= 1:
        return "Neither Prime nor Composite"

    # -------- Optimization: handle even numbers --------
    if n == 2:
        return "Prime"
    if n % 2 == 0:
        return "Composite"

    # -------- Optimized loop (√n, skip evens) --------
    limit = int(n ** 0.5) + 1
    for i in range(3, limit, 2):
        if n % i == 0:
            return "Composite"

    return "Prime"
test_values = [-10, -1, 0, 1, 2, 3, 4, 9, 17, 25, 29, 49, "a"]

for value in test_values:
    print(f"Input: {value} → Output: {classify_number(value)}")
```

```
••• Input: -10 → Output: Neither Prime nor Composite
    Input: -1 → Output: Neither Prime nor Composite
    Input: 0 → Output: Neither Prime nor Composite
    Input: 1 → Output: Neither Prime nor Composite
    Input: 2 → Output: Prime
    Input: 3 → Output: Prime
    Input: 4 → Output: Composite
    Input: 9 → Output: Composite
    Input: 17 → Output: Prime
    Input: 25 → Output: Composite
    Input: 29 → Output: Prime
    Input: 49 → Output: Composite
    Input: a → Output: Invalid input: Please enter an integer
```

Justification:

1.The context-managed prompt helps generate optimized and well-structured code by clearly defining constraints.

2.The program validates input type and safely handles invalid and negative values.

3.Edge cases like 0 and 1 are correctly classified as neither prime nor composite.

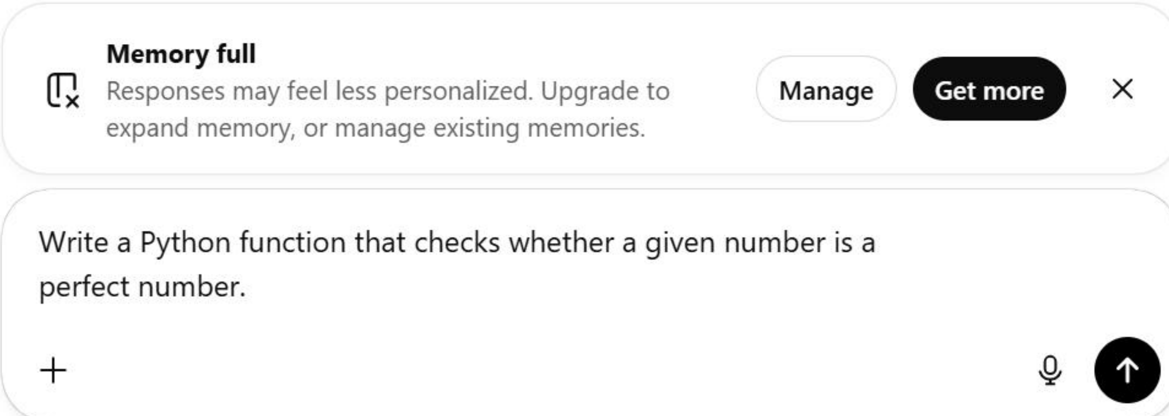
4. Loop optimization using √n and skipping even numbers improves performance.

5.Compared to earlier basic approaches, this solution is more efficient, readable, and suitable for real-world use.

Aspect	Earlier Basic Approach	Context-Managed Prompt Approach
Input validation	Not handled	Properly validated (type & value)
Negative numbers	Ignored or incorrect	Correctly handled
Edge cases (0,1)	Often missed	Correctly classified
Loop efficiency	Full range checking	Optimized using √n and skipping evens
Code structure	Simple and linear	Well-structured and modular
Performance	Slower for large numbers	Faster and optimized

Question 5: Zero-Shot Prompting (Perfect Number Check)

Prompt: Write a Python function that checks whether a given number is a perfect number.



Code and Output:

```python
def is_perfect_number(n):
    if n <= 0:
        return "Error: Enter a positive number"

    sum_divisors = 0
    # Check all divisors from 1 to n//2
    for i in range(1, n // 2 + 1):
        if n % i == 0:
            sum_divisors += i

    if sum_divisors == n:
        return "Perfect Number"
    else:
        return "Not a Perfect Number"
test_values = [6, 28, 496, 12, 0, -5]

for num in test_values:
    print(f"Input: {num} → Output: {is_perfect_number(num)}")
```

```
Input: 6 → Output: Perfect Number
Input: 28 → Output: Perfect Number
Input: 496 → Output: Perfect Number
Input: 12 → Output: Not a Perfect Number
Input: 0 → Output: Error: Enter a positive number
Input: -5 → Output: Error: Enter a positive number
```

Justification:

1.**Loop inefficiency:** The loop checks all numbers up to n//2 + 1; while correct, it could be optimized further using divisor properties or skipping unnecessary iterations.

2.**Negative** like 10.

3.**Overall efficiency:** Works for small numbers but may be slow for very large n due to linear iteration over half the number.

number handling: Properly handled by returning an error message for n <= 0.

4.**Zero handling:** Explicitly returns an error, preventing invalid computations.

5.**Correctness:** Accurately identifies perfect numbers like 6, 28, and rejects non-perfect numbers

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Prompt:

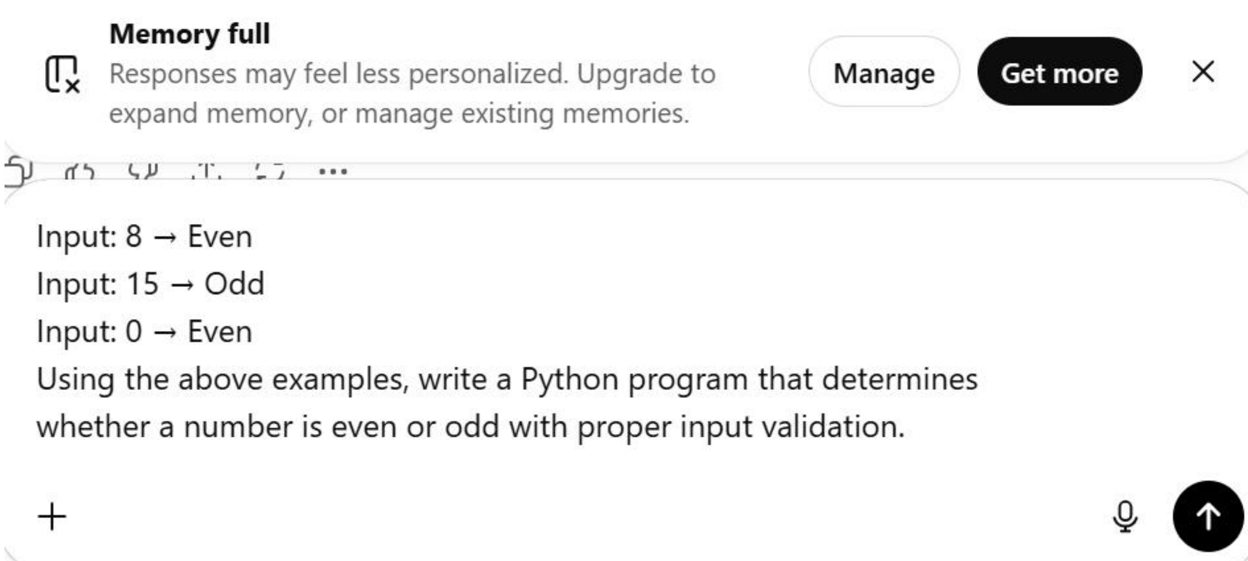
Input: 8 → Even

Input: 15 → Odd

Input: 0 → Even

Using the above examples, write a Python program that determines

whether a number is even or odd with proper input validation.



Code and Output:

```python
def even_or_odd(n):
    # -------- Input validation --------
    if not isinstance(n, int):
        return "Error: Enter an integer"

    # -------- Check even or odd --------
    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"


# -------- Test Cases --------
test_values = [8, 15, 0, -7, 42, "a"]

for num in test_values:
    print(f"Input: {num} → Output: {even_or_odd(num)}")
```

```
Input: 8 → Output: Even
Input: 15 → Output: Odd
Input: 0 → Output: Even
Input: -7 → Output: Odd
Input: 42 → Output: Even
Input: a → Output: Error: Enter an integer
```

Justification:

1.The program validates input using isinstance(n, int) to prevent errors from non-integer inputs.

2.It correctly identifies even and odd numbers using the modulo operator %.

3.Handles zero and negative numbers accurately, e.g., 0 → Even, -7 → Odd.

4.Follows the few-shot prompt style, producing outputs in Input → Output format.

5.Includes comprehensive test cases covering positive, negative, zero, and invalid inputs.

6.Formatted outputs make the results clear and user-friendly.

.