2303A51209
Batch:04
Assignment : 2.5
N.GOUTHAM

Task 1: Refactoring Odd/Even Logic (List Version)
❖ Scenario:
You are improving legacy code.
❖ Task:
Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.
❖ Expected Output:
❖ Original and improved code

Legacy code :

```
def calculate_sums(numbers):
    # Initialize variables
    odd_sum = 0
    even_sum = 0

    # Loop through the list
    for i in range(len(numbers)):
        num = numbers[i]
        # Check if number is even
        if num % 2 == 0:
            even_sum = even_sum + num
        else:
            odd_sum = odd_sum + num

    # Return results
    return odd_sum, even_sum

# Main execution
if __name__ == "__main__":
    # Test with a sample list
    my_list=[1,2,3,4,5,6,7,8,9, 10]

    result_odd,result_even=calculate_sums(my_list)

    print("Original List:", my_list)
    print("Sum of odd numbers:", result_odd)
    print("Sum of even numbers:", result_even)
```

Original List:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
Sum of odd numbers: 25
Sum of even numbers: 30
```

Improved Code :

```python
# Improved/Refactored Code - Sum of Odd and Even Numbers
# Refactored using AI for better readability, efficiency, and
maintainability

def calculate_sums(numbers):
    """
    Calculate the sum of odd and even numbers in a list.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd_numbers, sum_of_even_numbers)
    """
    # Use list comprehensions and sum() for more Pythonic code
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    even_sum = sum(num for num in numbers if num % 2 == 0)

    return odd_sum, even_sum


def calculate_sums_alternative(numbers):
    """
    Alternativeimplementationusingasinglepassthroughthelist.
    More efficient for large lists as it only iterates once.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd_numbers, sum_of_even_numbers)
    """
    odd_sum = 0
    even_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num  #Use+=operatorforcleanercode
        else:
            odd_sum += num

    return odd_sum, even_sum


def calculate_sums_with_validation(numbers):
    """
    Enhancedversionwithinputvalidationanderrorhandling.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd_numbers, sum_of_even_numbers)

    Raises:
        TypeError: If input is not a list
        ValueError: If list contains non-integer values
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

    if not all(isinstance(num, int) for num in numbers):
        raise ValueError("All elements must be integers")

    odd_sum = sum(num for num in numbers if num%2!=0)
    even_sum = sum(num for num in numbers if num%2==0)

    return odd_sum, even_sum


# Main execution
if __name__ == "__main__":
    # Test with a sample list
    my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    #Using the main function
    result_odd, result_even = calculate_sums(my_list)

    print("=" * 50)
    print("IMPROVED VERSION - Sum Calculator")
    print("=" * 50)
    print(f"Original List: {my_list}")
    print(f"Sum of odd numbers: {result_odd}")
    print(f"Sum of even numbers: {result_even}")
    print()

    #Demonstrate alternative implementation
    result_odd_alt, result_even_alt = calculate_sums_alternative(my_list)
    print("Alternative implementation results:")
    print(f"Sum of odd numbers: {result_odd_alt}")
    print(f"Sum of even numbers: {result_even_alt}")
    print()

    #Demonstrate validation version
    try:
        result_odd_val, result_even_val =
calculate_sums_with_validation(my_list)
        print("Validation version results:")
        print(f"Sum of odd numbers: {result_odd_val}")
        print(f"Sum of even numbers: {result_even_val}")
    except (TypeError, ValueError) as e:
        print(f"Error: {e}")
```

Output :

```
==================================================
IMPROVED VERSION - Sum Calculator
==================================================
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30

Alternative implementation results:
Sum of odd numbers: 25
```

Sum of even numbers: 30

Validation version results:
Sum of odd numbers: 25
Sum of even numbers: 30

Task 2: Area Calculation Explanation
❖ Scenario:
You are onboarding a junior developer.
❖ Task:
Ask Gemini to explain a function that calculates the area of different shapes.
❖ Expected Output:
➢ Code
➢ Explanation

Prompt : Im onboarding junior developer explain a function that calculates the area of different shapes.

Code :

```
def calculate_area(shape_type, **kwargs):
    """
    Calculates the area of different geometric shapes.

    Thisfunction takes the type of shape as an argument and then accepts
    additional keyword arguments specific to that shape's dimensions.

    Args:
        shape_type(str): The type of shape (e.g., 'circle', 'rectangle',
'triangle').
        **kwargs:Keyword arguments for the shape's dimensions.
                  -'circle': 'radius'
                  -'rectangle': 'length', 'width'
                  -'triangle': 'base', 'height'

    Returns:
        float:Thecalculated area of the shape.
        str:Anerrormessage if the shape type is unsupported or
arguments are missing.
    """

    if shape_type == 'circle':
        #Checkif'radius'isprovidedforacircle
        if 'radius' in kwargs:
            radius = kwargs['radius']
            #Theformulafortheareaofacircle is Pi * r^2
            import math
            return math.pi * (radius ** 2)
        else:
            return"Error:Circlerequires'radius' argument."

    elif shape_type == 'rectangle':
        #Checkif'length'and'width'areprovided for a rectangle
        if'length'inkwargsand'width'inkwargs:
            length = kwargs['length']
            width = kwargs['width']
            #Theformulafortheareaofarectangle is length * width
            return length * width
        else:
            return"Error:Rectanglerequires'length' and 'width'
arguments."

    elif shape_type == 'triangle':
        #Checkif'base'and'height'areprovided for a triangle
        if'base'inkwargsand'height'inkwargs:
            base = kwargs['base']
            height = kwargs['height']
            #Theformulafortheareaofatriangle is 0.5 * base*
height
            return 0.5 * base * height
        else:
            return"Error:Trianglerequires'base' and 'height'
arguments."

    else:
        #Handlecaseswheretheshapetypeisnot recognized
        return f"Error: Unsupported shape type '{shape_type}'."

# --- Examples of how to use the function ---
print(f"AreaofCircle(radius=5):{calculate_area('circle', radius=5)}")
print(f"Area of Rectangle (length=10, width=4):
{calculate_area('rectangle',length=10,width=4)}")
```

```
print(f"Area of Triangle (base=6, height=8): {calculate_area('triangle',
base=6, height=8)}")

#--- Examples of error handling ---
print(f"Missing argument for circle: {calculate_area('circle',
not_radius=5)}")
print(f"Unsupported shape: {calculate_area('square', side=5)}")
```

Output :

```
Area of Circle (radius=5): 78.53981633974483
Area of Rectangle (length=10, width=4): 40
Area of Triangle (base=6, height=8): 24.0
Missing argument for circle: Error: Circle requires 'radius' argument.
Unsupported shape: Error: Unsupported shape type 'square'.
```

Task 3: Prompt Sensitivity Experiment
❖ Scenario:
You are testing how AI responds to different prompts.
❖ Task:
Use Cursor AI with different prompts for the same problem and observe
code changes.
❖ Expected Output:
➢ Prompt list
➢ Code variations

Prompt : give me a function to check if the given string is palindrome or not

```
def is_palindrome(text):
    """
    Checkifagiven string is a palindrome.

    Args:
        text:String to check (case-insensitive, ignores spaces and punctuation)

    Returns:
        bool:True if the string is a palindrome, False otherwise

    Examples:
        >>>is_palindrome("racecar")
        True
        >>>is_palindrome("hello")
        False
        >>>is_palindrome("A man a plan a canal Panama")
        True
    """
    #Convert to lowercase and remove spaces for comparison
    cleaned_text = ''.join(text.lower().split())

    #Check if the cleaned string equals its reverse
    return cleaned_text == cleaned_text[::-1]
```

Prompt : Give me a function to check if the given string is palindrome or not use strict checks.

```
defis_palindrome_strict(text):
    """
    Strictpalindrome checker - only checks if the exact string is a palindrome.
    Case-sensitive and includes spaces.

    Args:
        text:String to check (case-sensitive, includes spaces)

    Returns:
        bool:True if the string is a palindrome, False otherwise

    Examples:
        >>>is_palindrome_strict("racecar")
        True
        >>>is_palindrome_strict("Racecar")
        False
        >>>is_palindrome_strict("a man a plan")
        False
    """
    returntext== text[::-1]
```

Prompt : Give me a function to check if the given string is palindrome or not use strict checks keep it advanced ignores back spaces,punctuation and case and only alphanumeric characters.

Code :

```
defis_palindrome_advanced(text):
    """
    Advanced palindrome checker that ignores spaces, punctuation, and case.
    Onlychecks alphanumeric characters.

    Args:
        text: String to check

    Returns:
        bool: True if the string is a palindrome, False otherwise
    Examples:
        >>> is_palindrome_advanced("A man, a plan, a canal: Panama")
        True
        >>> is_palindrome_advanced("race a car")
        False
    """
    #Remove all non-alphanumeric characters and convert to lowercase
    cleaned = ''.join(char.lower() for char in text if char.isalnum())

    return cleaned == cleaned[::-1]
```

Prompt: Give me a function to check if the given palindrome or not using two pointer technique(memory efficient) Ignore cases and spaces.

Code :

```
def is_palindrome_two_pointer(text):
    """
    Palindromecheckerusingtwo-pointer technique (more memory efficient).
    Ignores case and spaces.

    Args:
        text: String to check

    Returns:
        bool:Trueifthestringis a palindrome, False otherwise
    """
    #Cleanthetext:lowercaseand remove spaces
    cleaned=''.join(text.lower().split())

    # Two-pointer approach
    left = 0
    right = len(cleaned) - 1

    while left < right:
        ifcleaned[left]!=cleaned[right]:
            return False
        left += 1
        right -= 1

    return True
```

Outputs :
1. Basic Palindrome Checker (ignores case and spaces):
--
'racecar' -> True
'hello' -> False

```
'level' -> True
'madam' -> True
'A man a plan a canal Panama' -> True
'race a car' -> False
'A man, a plan, a canal: Panama' -> False
'12321' -> True
'12345' -> False
'' -> True
'a' -> True
'Aa' -> True


2. Strict Palindrome Checker (case-sensitive, includes spaces):
------------------------------------------------------------
'racecar' -> True
'hello' -> False
'level' -> True
'madam' -> True
'A man a plan a canal Panama' -> False
'race a car' -> False

3. Advanced Palindrome Checker (ignores punctuation):
------------------------------------------------------------
'racecar' -> True
'hello' -> False
'level' -> True
'madam' -> True
'A man a plan a canal Panama' -> True
'race a car' -> False
'A man, a plan, a canal: Panama' -> True
'12321' -> True
'12345' -> False
'' -> True
'a' -> True
'Aa' -> True


4. Two-Pointer Palindrome Checker (memory efficient):
------------------------------------------------------------
'racecar' -> True
'hello' -> False
'level' -> True
'madam' -> True
'A man a plan a canal Panama' -> True
'race a car' -> False
'A man, a plan, a canal: Panama' -> False
```

'12321' -> True
'12345' -> False
'' -> True
'a' -> True
'Aa' -> True

Task 4: ToolComparison Reflection

❖ Scenario:
You must recommend an AI coding tool.
❖ Task:
Based on your work in this topic, compare Gemini, Copilot, and Cursor AI
for usability and code quality.
❖ Expected Output:
Short written reflection

Written reflection :
I prefer using Copilot while working in VS Code and Gemini when collaborating with others, as both tools are extremely helpful during development. The auto-complete functionality in these tools significantly speeds up the coding process and reduces repetitive effort. The models they currently use are highly efficient and capable of generating clean, structured code that aligns well with our requirements. Another advantage is that they offer multiple approaches to solving the same problem, which helps in choosing the most optimal or readable solution. These tools also adapt well to different coding styles and project needs. Overall, I would strongly recommend these two AI tools for coding, as they are among the best options available today, especially for vibe coding where productivity and flow matter the most.