Prashun Lama

Student ID: 2359871

```
#Task1
import time

def convert_length(input_value, input_unit):
    """
    Convert length between meters and feet.
    :param input_value: Numeric value to be converted
    :param input_unit: The unit of the input value ('m' for meters, 'ft' for feet)
    :return: Converted value with appropriate unit
    """
    if input_unit == 'm':  # If the unit is meters
        return input_value * 3.28084, 'ft'  # Convert to feet
    elif input_unit == 'ft':  # If the unit is feet
        return input_value / 3.28084, 'm'  # Convert to meters
    else:
        raise ValueError("Unsupported length unit. Use 'm' for meters or 'ft' for feet.")

def convert_weight(input_value, input_unit):
    """
    Convert weight between kilograms and pounds.
    :param input_value: Numeric value to be converted
    :param input_unit: The unit of the input value ('kg' for kilograms, 'lbs' for pounds)
    :return: Converted value with appropriate unit
    """
    if input_unit == 'kg':  # If the unit is kilograms
        return input_value * 2.20462, 'lbs'  # Convert to pounds
    elif input_unit == 'lbs':  # If the unit is pounds
        return input_value / 2.20462, 'kg'  # Convert to kilograms
    else:
        raise ValueError("Unsupported weight unit. Use 'kg' for kilograms or 'lbs' for pounds.")

def convert_volume(input_value, input_unit):
    """
    Convert volume between liters and gallons.
    :param input_value: Numeric value to be converted
    :param input_unit: The unit of the input value ('L' for liters, 'gal' for gallons)
    :return: Converted value with appropriate unit
    """
    if input_unit == 'L':  # If the unit is liters
        return input_value * 0.264172, 'gal'  # Convert to gallons
    elif input_unit == 'gal':  # If the unit is gallons
        return input_value / 0.264172, 'L'  # Convert to liters
    else:
        raise ValueError("Unsupported volume unit. Use 'L' for liters or 'gal' for gallons.")

def main():
    """
```

```
        Main function to interact with the user and perform conversions based on user input.
        """
        print("Unit Converter: Choose a conversion type:")
        print("1. Length (meters to feet / feet to meters)")
        print("2. Weight (kilograms to pounds / pounds to kilograms)")
        print("3. Volume (liters to gallons / gallons to liters)")

        try:
            choice = int(input("Enter your choice (1-3): "))  # Get user choice for conversion type
            input_value = float(input("Enter the value to convert: "))  # Get the value to convert

            if choice == 1:  # If the user chooses length conversion
                input_unit = input("Enter the unit (m/ft): ")  # Get the unit for length (m or ft)
                converted_value, converted_unit = convert_length(input_value, input_unit)  # Perform the conversion
            elif choice == 2:  # If the user chooses weight conversion
                input_unit = input("Enter the unit (kg/lbs): ")  # Get the unit for weight (kg or lbs)
                converted_value, converted_unit = convert_weight(input_value, input_unit)  # Perform the conversion
            elif choice == 3:  # If the user chooses volume conversion
                input_unit = input("Enter the unit (L/gal): ")  # Get the unit for volume (L or gal)
                converted_value, converted_unit = convert_volume(input_value, input_unit)  # Perform the conversion
            else:
                print("Invalid choice. Please select a valid option.")  # Handle invalid choices
                return

            print(f"Converted Value: {converted_value:.2f} {converted_unit}")  # Output the converted value
        except ValueError as e:
            print(f"Error: {e}")  # Handle value errors (e.g., invalid number format)
        except Exception as e:
            print("An unexpected error occurred:", e)  # Handle other unexpected errors

if __name__ == "__main__":
    main()  # Run the main function
```

```
Unit Converter: Choose a conversion type:
1. Length (meters to feet / feet to meters)
2. Weight (kilograms to pounds / pounds to kilograms)
3. Volume (liters to gallons / gallons to liters)
Enter your choice (1-3): 2
Enter the value to convert: 78
Enter the unit (kg/lbs): kg
Converted Value: 171.96 lbs
```

```
#Task2
def calculate_sum(number_list):
    """Returns the sum of a list of numbers."""
    return sum(number_list)

def calculate_average(number_list):
    """Returns the average of a list of numbers."""
    return sum(number_list) / len(number_list) if number_list else 0

def find_maximum(number_list):
    """Returns the maximum value from a list of numbers."""
    return max(number_list)
```

```
def find_minimum(number_list):
    """Returns the minimum value from a list of numbers."""
    return min(number_list)

def main():
    """Main function to execute the mathematical operations program."""

    # Dictionary to map operation choices to their corresponding function
    operations = {
        '1': ('Sum', calculate_sum),
        '2': ('Average', calculate_average),
        '3': ('Maximum', find_maximum),
        '4': ('Minimum', find_minimum)
    }

    print("Choose an operation:")

    # Display the operation choices
    for key, (name, _) in operations.items():
        print(f"{key}. {name}")

    # Get user input for the chosen operation
    choice = input("Enter the number of the operation: ")

    # Validate the user's choice
    if choice not in operations:
        print("Invalid choice. Please select a valid option.")
        return

    try:
        # Get the list of numbers from the user
        number_list = list(map(float, input("Enter numbers separated by spaces: ").split()))

        # Validate if the list is not empty
        if not number_list:
            raise ValueError("The list cannot be empty.")

        # Fetch the operation name and function from the dictionary
        operation_name, operation_func = operations[choice]

        # Execute the chosen operation
        result = operation_func(number_list)

        # Output the result
        print(f"The {operation_name.lower()} of the numbers is: {result}")

    except ValueError as e:
        # Handle errors such as invalid input or empty list
        print(f"Error: {e}. Please enter valid numeric values.")

if __name__ == "__main__":
    main()  # Run the main function
```

```
Choose an operation:
1. Sum
```

```
2. Average
3. Maximum
4. Minimum
Enter the number of the operation: 1
Enter numbers separated by spaces: 55 23
The sum of the numbers is: 78.0
```

```
#Task3
def extract_every_other(input_list):
    """
    Extracts every other element from the input list, starting from the first element.

    Parameters:
    input_list (list): The input list from which elements are to be extracted.

    Returns:
    list: A new list containing every other element from the original list.
    """
    return input_list[::2]  # Use slicing with a step of 2 to select every second element

# Example usage
example_list = [1, 2, 3, 4, 5, 6]
result = extract_every_other(example_list)
print(result)
```

```
[1, 3, 5]
```

```
def get_sublist(input_list, start_index, end_index):
    """
    Returns a sublist from the given list, starting from the specified index and
    ending at another specified index (inclusive).

    Parameters:
    input_list (list): The list from which the sublist is to be extracted.
    start_index (int): The index where the sublist starts (inclusive).
    end_index (int): The index where the sublist ends (inclusive).

    Returns:
    list: A sublist containing elements from index start_index to end_index (inclusive).
    """
    return input_list[start_index:end_index + 1]  # Use slicing to include the end_index

# Example usage
example_list = [1, 2, 3, 4, 5, 6]
sublist = get_sublist(example_list, 2, 4)
print(sublist)
```

```
[3, 4, 5]
```

```
#Task5
def reverse_list(lst):
    """
    Reverses the given list using slicing.
```

```
    Parameters:
    lst (list): The input list to be reversed.

    Returns:
    list: A new list containing elements in reverse order.
    """
    return lst[::-1]  # Use slicing with a step of -1 to reverse the list

# Example usage
example_list = [1, 2, 3, 4, 5]
result = reverse_list(example_list)
print(result)
```

```
[5, 4, 3, 2, 1]
```

```
#Task-6
def remove_first_last(lst):
    """
    Removes the first and last elements of the given list and returns the resulting sublist.

    Parameters:
    lst (list): The input list from which the first and last elements are to be removed.

    Returns:
    list: A new list without the first and last elements.
    """
    return lst[1:-1]  # Slice from index 1 to the second-last element

# Example usage
example_list = [1, 2, 3, 4, 5]
result = remove_first_last(example_list)
print(result)  # Output: [2, 3, 4]
```

```
[2, 3, 4]
```

```
#Task-7
def get_first_n(lst, n):
    """
    Extracts the first n elements from the given list.

    Parameters:
    lst (list): The input list from which the first n elements are to be extracted.
    n (int): The number of elements to extract from the beginning of the list.

    Returns:
    list: A new list containing the first n elements of the original list.
    """
    return lst[:n]  # Slice to get the first n elements

# Example usage
example_list = [1, 2, 3, 4, 5]
result = get_first_n(example_list, 3)
```

```
print(result)  # Output: [1, 2, 3]
```

```
[1, 2, 3]
```

```
#Task8
def get_last_n(lst, n):
    """
    Extracts the last n elements from the given list.

    Parameters:
    lst (list): The input list from which the last n elements are to be extracted.
    n (int): The number of elements to extract from the end of the list.

    Returns:
    list: A new list containing the last n elements of the original list.
    """
    return lst[-n:]  # Slice to get the last n elements

# Example usage
example_list = [1, 2, 3, 4, 5]
result = get_last_n(example_list, 2)
print(result)  # Output: [4, 5]
```

```
[4, 5]
```

```
#Task- 9
def reverse_skip(lst):
    """
    Extracts elements in reverse order starting from the second-to-last element,
    skipping one element in between.

    Parameters:
    lst (list): The input list from which elements are to be extracted.

    Returns:
    list: A new list containing every second element starting from the second-to-last element, moving backward.
    """
    return lst[-2::-2]  # Slice to start from second-to-last and skip every second element backward

# Example usage
example_list = [1, 2, 3, 4, 5, 6]
result = reverse_skip(example_list)
print(result)
```

```
[5, 3, 1]
```

```
#Task-10
def flatten(lst):
    """
    Flattens a nested list into a single-dimensional list.
```

```
    Parameters:
    lst (list): The input nested list containing sublists.

    Returns:
    list: A new list with all elements in a single dimension.
    """
    flat_list = []  # Initialize an empty list to store flattened elements
    for sublist in lst:
        flat_list.extend(sublist)  # Extend the list by adding elements from each sublist
    return flat_list

# Example usage
example_list = [[1, 2], [3, 4], [5]]
result = flatten(example_list)
print(result)
```

```
[1, 2, 3, 4, 5]
```

```
#Task-11
def access_nested_element(lst, indices):
    """
    Extracts a specific element from a nested list given a list of indices.

    Parameters:
    lst (list): The nested list from which to extract the element.
    indices (list): A list of indices representing the path to the desired element.

    Returns:
    any: The element at the specified indices, or None if indices are invalid.
    """
    try:
        element = lst  # Start with the original nested list
        for index in indices:
            element = element[index]  # Navigate deeper using the provided indices
        return element
    except (IndexError, TypeError):
        return None  # Return None if indices are out of range or invalid

# Example usage
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
result = access_nested_element(nested_list, [1, 2])
print(result)
```

```
6
```

```
#Task-12
def sum_nested(lst):
    """
    Recursively calculates the sum of all numbers in a nested list.

    Parameters:
    lst (list): The nested list containing integers or sublists.
```

```
    Returns:
    int: The sum of all numbers in the nested list.
    """
    total = 0
    for element in lst:
        if isinstance(element, list):  # If it's a list, recurse into it
            total += sum_nested(element)
        else:  # If it's an integer, add it to the total
            total += element
    return total

# Example usage
nested_list = [[1, 2], [3, [4, 5]], 6]
result = sum_nested(nested_list)
print(result)
```

```
21
```

```
#Task-13
def remove_element(lst, elem):
    """
    Recursively removes all occurrences of a specific element from a nested list.

    Parameters:
    lst (list): The nested list from which the element should be removed.
    elem (any): The element to remove from the list.

    Returns:
    list: A new nested list with all occurrences of the element removed.
    """
    result = []
    for item in lst:
        if isinstance(item, list):  # If the item is a list, recurse into it
            new_sublist = remove_element(item, elem)
            result.append(new_sublist)  # Append modified sublist
        elif item != elem:  # Only add elements that are not equal to `elem`
            result.append(item)
    return result

# Example usage
nested_list = [[1, 2], [3, 2], [4, 5]]
result = remove_element(nested_list, 2)
print(result)
```

```
[[1], [3], [4, 5]]
```

```
#Task-14
def find_max(lst):
    """
    Recursively finds the maximum element in a nested list.

    Parameters:
    lst (list): The nested list containing integers or sublists.
```

```
    Returns:
    int: The maximum element in the nested list.
    """
    max_value = float('-inf')  # Initialize max as negative infinity

    for item in lst:
        if isinstance(item, list):  # If item is a list, recurse into it
            max_value = max(max_value, find_max(item))
        else:  # If item is a number, compare it with max_value
            max_value = max(max_value, item)

    return max_value

# Example usage
nested_list = [[1, 2], [3, [4, 5]], 6]
result = find_max(nested_list)
print(result)
```

```
6
```

```
#Task-15
def count_occurrences(lst, elem):
    """
    Recursively counts the occurrences of a specific element in a nested list.

    Parameters:
    lst (list): The nested list to search in.
    elem (any): The element whose occurrences need to be counted.

    Returns:
    int: The number of times elem appears in the nested list.
    """
    count = 0
    for item in lst:
        if isinstance(item, list):  # If item is a list, recurse into it
            count += count_occurrences(item, elem)
        elif item == elem:  # If item matches elem, increase count
            count += 1
    return count

# Example usage
nested_list = [[1, 2], [2, 3], [2, 4]]
result = count_occurrences(nested_list, 2)
print(result)
```

```
3
```

```
#Task16
def deep_flatten(lst):
    """
    Recursively flattens a deeply nested list into a single list.
```

```
        Parameters:
        lst (list): The nested list to be flattened.

        Returns:
        list: A flattened version of the input list.
        """
        flattened_list = []
        for item in lst:
            if isinstance(item, list):  # If the item is a list, recurse into it
                flattened_list.extend(deep_flatten(item))
            else:  # If it's not a list, add it to the result
                flattened_list.append(item)
        return flattened_list

    # Example usage
    nested_list = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
    result = deep_flatten(nested_list)
    print(result)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
    #Task-17
    def average_nested(lst):
        """
        Recursively calculates the average of all elements in a nested list.

        Parameters:
        lst (list): The nested list containing numbers.

        Returns:
        float: The average of all numbers in the nested list.
        """
        def flatten_and_sum(lst):
            """ Helper function to flatten list and sum elements with count. """
            total_sum = 0
            count = 0
            for item in lst:
                if isinstance(item, list):
                    sub_sum, sub_count = flatten_and_sum(item)
                    total_sum += sub_sum
                    count += sub_count
                else:
                    total_sum += item
                    count += 1
            return total_sum, count

        total, count = flatten_and_sum(lst)
        return total / count if count > 0 else 0  # Avoid division by zero

    # Example usage
    nested_list = [[1, 2], [3, 4], [5, 6]]
    result = average_nested(nested_list)
    print(result)
```

```
3.5
```

NUMPY

```
import numpy as np

def create_empty_array():
    """
    Creates and returns an empty 2x2 NumPy array.

    Returns:
    numpy.ndarray: A 2x2 uninitialized (empty) array.
    """
    return np.empty((2, 2))  # Create a 2x2 uninitialized array

def create_ones_array():
    """
    Creates and returns a 4x2 NumPy array filled with ones.

    Returns:
    numpy.ndarray: A 4x2 array filled with ones.
    """
    return np.ones((4, 2))  # Create a 4x2 array filled with ones

def create_filled_array(array_shape, fill_value):
    """
    Creates and returns a NumPy array of a given shape, filled with a specific value.

    Parameters:
    array_shape (tuple): Shape of the array.
    fill_value (int/float): Value to fill the array with.

    Returns:
    numpy.ndarray: An array filled with the specified value.
    """
    return np.full(array_shape, fill_value)  # Create an array filled with the specified value

def create_zeros_like(reference_array):
    """
    Creates and returns a NumPy array of zeros with the same shape and type as a given array.

    Parameters:
    reference_array (numpy.ndarray): Reference array.

    Returns:
    numpy.ndarray: A zero-filled array with the same shape and type as reference_array.
    """
    return np.zeros_like(reference_array)  # Create an array of zeros with the same shape as reference_array

def create_ones_like(reference_array):
    """
    Creates and returns a NumPy array of ones with the same shape and type as a given array.
```

```
        Parameters:
        reference_array (numpy.ndarray): Reference array.

        Returns:
        numpy.ndarray: A ones-filled array with the same shape and type as reference_array.
        """
        return np.ones_like(reference_array)  # Create an array of ones with the same shape as reference_array

    def convert_list_to_numpy(py_list):
        """
        Converts a Python list into a NumPy array.

        Parameters:
        py_list (list): List to convert.

        Returns:
        numpy.ndarray: NumPy array containing the list elements.
        """
        return np.array(py_list)  # Convert the Python list to a NumPy array

    # Example Usage
    empty_array = create_empty_array()
    print("Empty Array (2x2):\n", empty_array)

    ones_array = create_ones_array()
    print("\nAll Ones Array (4x2):\n", ones_array)

    filled_array = create_filled_array((3, 3), 7)  # Example: Filling with 7
    print("\nArray Filled with 7 (3x3):\n", filled_array)

    reference_array = np.array([[5, 6, 7], [8, 9, 10]])

    zeros_like_array = create_zeros_like(reference_array)
    print("\nZeros Array with Same Shape as Reference:\n", zeros_like_array)

    ones_like_array = create_ones_like(reference_array)
    print("\nOnes Array with Same Shape as Reference:\n", ones_like_array)

    new_list = [1, 2, 3, 4]
    numpy_array = convert_list_to_numpy(new_list)
    print("\nConverted NumPy Array:\n", numpy_array)
```

```
Empty Array (2x2):
 [[1.14e-322 2.52e-322]
 [6.42e-323 1.43e-322]]

All Ones Array (4x2):
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]

Array Filled with 7 (3x3):
 [[7 7 7]
 [7 7 7]
 [7 7 7]]
```

```
 Zeros Array with Same Shape as Reference:
  [[0 0 0]
  [0 0 0]]

 Ones Array with Same Shape as Reference:
  [[1 1 1]
  [1 1 1]]

 Converted NumPy Array:
  [1 2 3 4]
```

```
#Num-py Problem Number-2
import numpy as np

def create_range_array():
    """
    Creates and returns an array with values ranging from 10 to 49.

    Returns:
    numpy.ndarray: Array with values from 10 to 49.
    """
    return np.arange(10, 50)

def create_3x3_matrix():
    """
    Creates and returns a 3x3 matrix with values ranging from 0 to 8.

    Returns:
    numpy.ndarray: A 3x3 matrix with values from 0 to 8.
    """
    return np.arange(9).reshape(3, 3)

def create_identity_matrix():
    """
    Creates and returns a 3x3 identity matrix.

    Returns:
    numpy.ndarray: A 3x3 identity matrix.
    """
    return np.eye(3)

def create_random_array_mean():
    """
    Creates a random array of size 30 and calculates its mean.

    Returns:
    tuple: (random array, mean value)
    """
    random_array = np.random.random(30)  # Generate random values
    mean_value = random_array.mean()  # Compute the mean
    return random_array, mean_value

def create_10x10_random_min_max():
    """
    Creates a 10x10 matrix with random values and finds the minimum and maximum values.
```

```
        Returns:
        tuple: (random 10x10 matrix, min value, max value)
        """
        random_matrix = np.random.random((10, 10))  # Generate a 10x10 matrix with random values
        min_value = random_matrix.min()  # Find the minimum value
        max_value = random_matrix.max()  # Find the maximum value
        return random_matrix, min_value, max_value

    def replace_fifth_element():
        """
        Creates a zero array of size 10 and replaces the 5th element with 1.

        Returns:
        numpy.ndarray: A modified array with the 5th element set to 1.
        """
        arr = np.zeros(10)  # Create a zero array
        arr[4] = 1  # Replace the 5th element (index 4) with 1
        return arr

    def reverse_array():
        """
        Reverses a given array.

        Returns:
        numpy.ndarray: A reversed array.
        """
        arr = np.array([1, 2, 0, 0, 4, 0])
        return arr[::-1]  # Reverse using slicing

    def create_border_array():
        """
        Creates a 2D array with 1 on the border and 0 inside.

        Returns:
        numpy.ndarray: A 2D array with border 1s and inside 0s.
        """
        arr = np.ones((5, 5))  # Create an array of ones
        arr[1:-1, 1:-1] = 0  # Set inner elements to 0
        return arr

    def create_checkerboard_pattern():
        """
        Creates an 8x8 matrix and fills it with a checkerboard pattern.

        Returns:
        numpy.ndarray: An 8x8 checkerboard matrix.
        """
        checkerboard = np.zeros((8, 8), dtype=int)  # Create an 8x8 zero matrix
        checkerboard[1::2, ::2] = 1  # Fill alternating rows with 1s
        checkerboard[::2, 1::2] = 1  # Fill alternating columns with 1s
        return checkerboard

    # Example Usage
    print("Array with values from 10 to 49:\n", create_range_array())
```

```
print("\n3x3 Matrix with values 0 to 8:\n", create_3x3_matrix())

print("\n3x3 Identity Matrix:\n", create_identity_matrix())

random_array, mean_value = create_random_array_mean()
print("\nRandom Array of Size 30:\n", random_array)
print("Mean Value:", mean_value)

random_matrix, min_value, max_value = create_10x10_random_min_max()
print("\n10x10 Random Matrix:\n", random_matrix)
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)

print("\nZero Array with 5th Element as 1:\n", replace_fifth_element())

print("\nReversed Array:\n", reverse_array())

print("\n2D Array with Border 1s and Inside 0s:\n", create_border_array())

print("\n8x8 Checkerboard Pattern:\n", create_checkerboard_pattern())
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Random Array of Size 30:
 [0.92656556 0.25436145 0.90665149 0.95513996 0.03991911 0.41529812
 0.31134429 0.20026827 0.24680429 0.44917628 0.92269371 0.6187773
 0.10440382 0.35700936 0.07786696 0.46851191 0.50892127 0.53177241
 0.57822852 0.96040591 0.97154568 0.84567428 0.31628695 0.20753129
 0.83455669 0.75081642 0.91244985 0.46767542 0.16283958 0.02706412]
Mean Value: 0.5110186761542713

10x10 Random Matrix:
 [[0.25885671 0.34181359 0.35413112 0.19979999 0.07687198 0.69173352
  0.67895251 0.19429422 0.13327756 0.26424078]
 [0.02925492 0.09288418 0.55641551 0.33388838 0.55443987 0.79084826
  0.27258885 0.21511268 0.53178375 0.00778841]
 [0.01877623 0.39919077 0.24391574 0.76218117 0.36693431 0.3346853
  0.26323158 0.94028524 0.10408292 0.48686522]
 [0.7956814  0.58844598 0.41081018 0.18930509 0.60223651 0.45246536
  0.54556669 0.56530993 0.52352977 0.46080279]
 [0.68579414 0.62965493 0.22465719 0.31965249 0.24025181 0.31603225
  0.52645073 0.72997264 0.54047811 0.00624176]
 [0.99991819 0.90668124 0.81448749 0.59439783 0.17722456 0.03092374
  0.51032284 0.90838998 0.63390224 0.67594136]
 [0.38922168 0.40315318 0.37175609 0.99343183 0.44347533 0.85847952
  0.64675187 0.5774483  0.83795619 0.28097715]
```

```
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

Reversed Array:
 [0 4 0 0 2 1]

2D Array with Border 1s and Inside 0s:
 [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]

8x8 Checkerboard Pattern:
 [[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

```
#Num-py Problem Number-3
import numpy as np

# Define the arrays
x = np.array([[1, 2], [3, 5]])  # A 2x2 array
y = np.array([[5, 6], [7, 8]])  # Another 2x2 array
v = np.array([9, 10])  # A 1D array of size 2
w = np.array([11, 12])  # Another 1D array of size 2

# 1. Add the two arrays
def add_arrays(x, y):
    """
    Adds two 2D numpy arrays element-wise.

    Args:
    x (numpy.ndarray): First 2D array.
    y (numpy.ndarray): Second 2D array.

    Returns:
    numpy.ndarray: Resultant array from element-wise addition of x and y.
    """
    return x + y

add_result = add_arrays(x, y)
print("Addition of x and y:\n", add_result)

# 2. Subtract the two arrays
def subtract_arrays(x, y):
    """
    Subtracts the second array (y) from the first array (x) element-wise.

    Args:
    x (numpy.ndarray): First 2D array.
    y (numpy.ndarray): Second 2D array.

    Returns:
```

```
        numpy.ndarray: Resultant array from element-wise subtraction of x and y.
        """
        return x - y

    sub_result = subtract_arrays(x, y)
    print("\nSubtraction of x and y:\n", sub_result)

    # 3. Multiply the array with any integers of your choice (let's use 2 for demonstration)
    def multiply_array(x, scalar):
        """
        Multiplies each element of the 2D numpy array x by a scalar value.

        Args:
        x (numpy.ndarray): A 2D numpy array.
        scalar (int): Scalar value to multiply each element of the array by.

        Returns:
        numpy.ndarray: A new numpy array with elements multiplied by scalar.
        """
        return x * scalar

    mul_result = multiply_array(x, 2)
    print("\nMultiplying x by 2:\n", mul_result)

    # 4. Find the square of each element of the array
    def square_elements(x):
        """
        Computes the square of each element in the 2D numpy array x.

        Args:
        x (numpy.ndarray): A 2D numpy array.

        Returns:
        numpy.ndarray: A new array with each element squared.
        """
        return np.square(x)

    square_result = square_elements(x)
    print("\nSquare of each element in x:\n", square_result)

    # 5. Find the dot product between: v and w, x and v, x and y
    def dot_product(a, b):
        """
        Computes the dot product between two numpy arrays.

        Args:
        a (numpy.ndarray): First array.
        b (numpy.ndarray): Second array.

        Returns:
        int or numpy.ndarray: Dot product of a and b.
        """
        return np.dot(a, b)

    # v and w (1D arrays)
    dot_vw = dot_product(v, w)
```

```
print("\nDot product between v and w:", dot_vw)

# x and v (2D and 1D arrays)
dot_xv = dot_product(x, v)
print("\nDot product between x and v:\n", dot_xv)

# x and y (2D arrays)
dot_xy = dot_product(x, y)
print("\nDot product between x and y:\n", dot_xy)

# 6. Concatenate x and y along row (axis=0), and v and w along columns (axis=1)
def concatenate_arrays(x, y, axis):
    """
    Concatenates two numpy arrays along the specified axis.

    Args:
    x (numpy.ndarray): First array.
    y (numpy.ndarray): Second array.
    axis (int): Axis along which to concatenate (0 for rows, 1 for columns).

    Returns:
    numpy.ndarray: Concatenated array.
    """
    return np.concatenate((x, y), axis=axis)

# Concatenate x and y along row (axis=0)
concat_xy_row = concatenate_arrays(x, y, axis=0)
print("\nConcatenating x and y along row:\n", concat_xy_row)

# Concatenate v and w along column (axis=1) - reshaping v and w to be 2D
concat_vw_col = np.concatenate((v.reshape(-1, 1), w.reshape(-1, 1)), axis=1)
print("\nConcatenating v and w along column:\n", concat_vw_col)

# 7. Concatenate x and v (observe and explain the error)
def try_concatenate_x_and_v(x, v):
    """
    Tries to concatenate x (2D array) and v (1D array) along columns (axis=1).
    If it fails, the error is caught and returned.

    Args:
    x (numpy.ndarray): A 2D numpy array.
    v (numpy.ndarray): A 1D numpy array.

    Returns:
    str: Error message if concatenation fails.
    """
    try:
        # Attempting to concatenate x and v along columns (axis=1)
        return np.concatenate((x, v), axis=1)
    except Exception as e:
        return str(e)

# Trying to concatenate x and v
error_message = try_concatenate_x_and_v(x, v)
print("\nError when trying to concatenate x and v:", error_message)
```

```
Addition of x and y:
 [[ 6  8]
 [10 13]]

Subtraction of x and y:
 [[-4 -4]
 [-4 -3]]

Multiplying x by 2:
 [[ 2  4]
 [ 6 10]]

Square of each element in x:
 [[ 1  4]
 [ 9 25]]

Dot product between v and w: 219

Dot product between x and v:
 [29 77]

Dot product between x and y:
 [[19 22]
 [50 58]]

Concatenating x and y along row:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]

Concatenating v and w along column:
 [[ 9 11]
 [10 12]]

Error when trying to concatenate x and v: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index 1 has 1 c
```

```
#Num-py Problem Number-4


# Define the arrays
A = np.array([[3, 4], [7, 8]])  # 2x2 matrix A
B = np.array([[5, 3], [2, 1]])  # 2x2 matrix B

# 1. Prove A.A⁻¹ = I (Identity Matrix)
def prove_identity(A):
    """
    Proves that A * A⁻¹ = Identity matrix.

    Args:
    A (numpy.ndarray): A square matrix.

    Returns:
    numpy.ndarray: Identity matrix.
    """
    A_inv = np.linalg.inv(A)  # Inverse of A
```

```
        I = np.dot(A, A_inv)  # Matrix multiplication of A and A⁻¹
        return I

    identity_matrix = prove_identity(A)
    print("A * A⁻¹ (Identity Matrix):\n", identity_matrix)

    # 2. Prove AB ≠ BA (Matrix multiplication is not commutative)
    def prove_non_commutative(A, B):
        """
        Proves that AB ≠ BA.

        Args:
        A (numpy.ndarray): Matrix A.
        B (numpy.ndarray): Matrix B.

        Returns:
        tuple: AB and BA results.
        """
        AB = np.dot(A, B)
        BA = np.dot(B, A)
        return AB, BA

    AB, BA = prove_non_commutative(A, B)
    print("\nAB = \n", AB)
    print("BA = \n", BA)
    print("\nAB ≠ BA: ", not np.array_equal(AB, BA))

    # 3. Prove (AB)ᵀ = BᵀAᵀ (Transpose of a product)
    def prove_transpose_product(A, B):
        """
        Proves that (AB)ᵀ = BᵀAᵀ.

        Args:
        A (numpy.ndarray): Matrix A.
        B (numpy.ndarray): Matrix B.

        Returns:
        tuple: (AB)ᵀ and BᵀAᵀ results.
        """
        AB_transpose = np.transpose(np.dot(A, B))
        B_transpose_A_transpose = np.dot(np.transpose(B), np.transpose(A))
        return AB_transpose, B_transpose_A_transpose

    AB_transpose, B_transpose_A_transpose = prove_transpose_product(A, B)
    print("\n(AB)ᵀ = \n", AB_transpose)
    print("BᵀAᵀ = \n", B_transpose_A_transpose)
    print("\n(AB)ᵀ = BᵀAᵀ: ", np.array_equal(AB_transpose, B_transpose_A_transpose))

    # 4. Solve the system of linear equations using Inverse Method
    # System of equations:
    # 2x - 3y + z = -1
    # x - y + 2z = -3
    # 3x + y - z = 9

    # Represent the system in matrix form: AX = B
    A_matrix = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])  # Coefficient matrix A
```

```
B_matrix = np.array([-1, -3, 9])  # Constants vector B

# Solve for X using inverse method: X = A⁻¹ * B
def solve_linear_system(A_matrix, B_matrix):
    """
    Solves the linear system of equations using inverse method.

    Args:
    A_matrix (numpy.ndarray): Coefficient matrix.
    B_matrix (numpy.ndarray): Constants vector.

    Returns:
    numpy.ndarray: Solution vector X (values for x, y, z).
    """
    A_inv = np.linalg.inv(A_matrix)  # Inverse of matrix A
    X = np.dot(A_inv, B_matrix)  # Solving for X
    return X

solution = solve_linear_system(A_matrix, B_matrix)
print("\nSolution for the system of equations (using inverse method):", solution)

# 5. Solve the system of linear equations using np.linalg.inv function directly
def solve_using_linalg(A_matrix, B_matrix):
    """
    Solves the linear system of equations using np.linalg.inv directly.

    Args:
    A_matrix (numpy.ndarray): Coefficient matrix.
    B_matrix (numpy.ndarray): Constants vector.

    Returns:
    numpy.ndarray: Solution vector X.
    """
    X_linalg = np.linalg.solve(A_matrix, B_matrix)  # Using np.linalg.solve to solve AX = B
    return X_linalg

solution_linalg = solve_using_linalg(A_matrix, B_matrix)
print("\nSolution using np.linalg.solve:", solution_linalg)
```

```
A * A⁻¹ (Identity Matrix):
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]

AB =
 [[23 13]
 [51 29]]
BA =
 [[36 44]
 [13 16]]

AB ≠ BA:  True

(AB)ᵀ =
 [[23 51]
 [13 29]]
BᵀAᵀ =
```

```
 [[23 51]
 [13 29]]

(AB)ᵀ = BᵀAᵀ:  True

Solution for the system of equations (using inverse method): [ 2.  1. -2.]

Solution using np.linalg.solve: [ 2.  1. -2.]
```

```
import numpy as np

# Define the matrices
matrix_A = np.array([[3, 4], [7, 8]])  # 2x2 matrix A
matrix_B = np.array([[5, 3], [2, 1]])  # 2x2 matrix B

# 1. Prove A * A⁻¹ = I (Identity Matrix)
def prove_identity(matrix_A):
    """
    Proves that A * A⁻¹ = Identity matrix.

    Args:
    matrix_A (numpy.ndarray): A square matrix.

    Returns:
    numpy.ndarray: Identity matrix.
    """
    A_inv = np.linalg.inv(matrix_A)  # Inverse of matrix_A
    identity_matrix = np.dot(matrix_A, A_inv)  # Matrix multiplication of A and A⁻¹
    return identity_matrix

identity_matrix = prove_identity(matrix_A)
print("A * A⁻¹ (Identity Matrix):\n", identity_matrix)

# 2. Prove AB ≠ BA (Matrix multiplication is not commutative)
def prove_non_commutative(matrix_A, matrix_B):
    """
    Proves that AB ≠ BA.

    Args:
    matrix_A (numpy.ndarray): Matrix A.
    matrix_B (numpy.ndarray): Matrix B.

    Returns:
    tuple: AB and BA results.
    """
    AB = np.dot(matrix_A, matrix_B)
    BA = np.dot(matrix_B, matrix_A)
    return AB, BA

AB, BA = prove_non_commutative(matrix_A, matrix_B)
print("\nAB = \n", AB)
print("BA = \n", BA)
print("\nAB ≠ BA: ", not np.array_equal(AB, BA))

# 3. Prove (AB)ᵀ = BᵀAᵀ (Transpose of a product)
def prove_transpose_product(matrix_A, matrix_B):
```

```
        """
        Proves that (AB)ᵀ = BᵀAᵀ.

        Args:
        matrix_A (numpy.ndarray): Matrix A.
        matrix_B (numpy.ndarray): Matrix B.

        Returns:
        tuple: (AB)ᵀ and BᵀAᵀ results.
        """
        AB_transpose = np.transpose(np.dot(matrix_A, matrix_B))
        B_transpose_A_transpose = np.dot(np.transpose(matrix_B), np.transpose(matrix_A))
        return AB_transpose, B_transpose_A_transpose

    AB_transpose, B_transpose_A_transpose = prove_transpose_product(matrix_A, matrix_B)
    print("\n(AB)ᵀ = \n", AB_transpose)
    print("BᵀAᵀ = \n", B_transpose_A_transpose)
    print("\n(AB)ᵀ = BᵀAᵀ: ", np.array_equal(AB_transpose, B_transpose_A_transpose))

    # 4. Solve the system of linear equations using Inverse Method
    # System of equations:
    # 2x - 3y + z = -1
    # x - y + 2z = -3
    # 3x + y - z = 9

    # Represent the system in matrix form: AX = B
    coefficient_matrix = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])  # Coefficient matrix A
    constant_vector = np.array([-1, -3, 9])  # Constants vector B

    # Solve for X using inverse method: X = A⁻¹ * B
    def solve_linear_system(coefficient_matrix, constant_vector):
        """
        Solves the linear system of equations using the inverse method.

        Args:
        coefficient_matrix (numpy.ndarray): Coefficient matrix.
        constant_vector (numpy.ndarray): Constants vector.

        Returns:
        numpy.ndarray: Solution vector X (values for x, y, z).
        """
        A_inv = np.linalg.inv(coefficient_matrix)  # Inverse of coefficient_matrix
        solution_vector = np.dot(A_inv, constant_vector)  # Solving for X
        return solution_vector

    solution = solve_linear_system(coefficient_matrix, constant_vector)
    print("\nSolution for the system of equations (using inverse method):", solution)

    # 5. Solve the system of linear equations using np.linalg.solve function directly
    def solve_using_linalg(coefficient_matrix, constant_vector):
        """
        Solves the linear system of equations using np.linalg.solve directly.

        Args:
        coefficient_matrix (numpy.ndarray): Coefficient matrix.
        constant_vector (numpy.ndarray): Constants vector.
```

```
    Returns:
    numpy.ndarray: Solution vector X.
    """
    solution_vector_linalg = np.linalg.solve(coefficient_matrix, constant_vector)  # Using np.linalg.solve to solve AX = B
    return solution_vector_linalg

solution_linalg = solve_using_linalg(coefficient_matrix, constant_vector)
print("\nSolution using np.linalg.solve:", solution_linalg)
```

```
A * A⁻¹ (Identity Matrix):
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]

AB =
 [[23 13]
 [51 29]]
BA =
 [[36 44]
 [13 16]]

AB ≠ BA:  True

(AB)ᵀ =
 [[23 51]
 [13 29]]
BᵀAᵀ =
 [[23 51]
 [13 29]]

(AB)ᵀ = BᵀAᵀ:  True

Solution for the system of equations (using inverse method): [ 2.  1. -2.]

Solution using np.linalg.solve: [ 2.  1. -2.]
```