```
Choose an operation:
1. Sum
2. Average
3. Maximum
4. Minimum
Enter the number of the operation:  2
Enter numbers separated by spaces:  56 66
The average of the numbers is: 61.0
```

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

    Parameters:
    lst (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 lst[::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)  # Output: [1, 3, 5]
```

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

```python
#Task 4
def get_sublist(lst, start, end):
    """
    Returns a sublist from the given list, starting from the specified index and
    ending at another specified index (inclusive).

    Parameters:
    lst (list): The input list from which a sublist is to be extracted.
    start (int): The starting index of the sublist.
    end (int): The ending index of the sublist (inclusive).

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

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

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

```python
#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 r

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

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

```python
#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]
```

```
Click to add a breakpoint
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
    n (int): The number of elements to extract from the beginn

    Returns:
    list: A new list containing the first n elements of the or
    """
    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]
```

```python
#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]
```

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

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

    Returns:
    list: A new list containing every second element startir
    """
    return lst[-2::-2]  # Slice to start from second-to-last

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

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

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

    Parameters:
    lst (list): The input nested list containing sublist

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

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

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

```python
#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
```

```python
#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 in
            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
```

```python
def remove_element(lst, elem):
    """
    Recursively removes all occurrences of a specific element from a nested lis

    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]]
```

```python
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 int
            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
```

```python
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
```

```python
def deep_flatten(lst):
    """
    Recursively flattens a deeply nested list into a single li

    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, r
            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]
```

```python
#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
```

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

```
3]

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

```python
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
```

```python
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
```

```python
#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]
```

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

    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 wi
        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)
```

38]

3.5

```python
#Num-py Problem Number-1
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))

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))
```

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

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

    Returns:
    numpy.ndarray: An array filled with the specified value.
    """
    return np.full(shape, 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)
```

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

    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)

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

    Parameters:
    new_list (list): List to convert.

    Returns:
    numpy.ndarray: NumPy array containing the list elements.
    """
    return np.array(new_list)
```

```
Click to add a breakpoint
# 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):
```

```
Empty Array (2x2):
 [[6.23042070e-307 4.67296746e-307]
 [1.69121096e-306 1.06736388e-311]]

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]
```

```
Array with values from 10 to 49:
 [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]

3x3 Matrix with values 0 to 8:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]

3x3 Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Random Array of Size 30:
 [0.63538081 0.14673626 0.68950645 0.8155931  0.73937707 0.32251608
 0.89140924 0.39799551 0.17924033 0.47361002 0.9729675  0.85004997
 0.73127987 0.41065269 0.79829463 0.13928692 0.18433471 0.72461772
 0.91881735 0.24286517 0.9173989  0.71128722 0.79748143 0.83691129
 0.20793815 0.53294832 0.83054932 0.91680279 0.54942954 0.25988337]
Mean Value: 0.5941720572710947

 Random Array of Size 30:
  [0.63538081 0.14673626 0.68950645 0.8155931  0.73937707 0.32251608
  0.89140924 0.39799551 0.17924033 0.47361002 0.9729675  0.85004997
  0.73127987 0.41065269 0.79829463 0.13928692 0.18433471 0.72461772
  0.91881735 0.24286517 0.9173989  0.71128722 0.79748143 0.83691129
  0.20793815 0.53294832 0.83054932 0.91680279 0.54942954 0.25988337]
 Mean Value: 0.5941720572710947

 10x10 Random Matrix:
  [[0.08386685 0.11933945 0.36968344 0.05124527 0.56882162 0.11308989
   0.57989476 0.61077115 0.56369559 0.04245297]
 ...
  [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]]
```

```
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]]
...
 [[ 9 11]
 [10 12]]
```

```
A * A⁻¹ (Identity Matrix):
 [[1.0000000e+00 4.4408921e-16]
 [0.0000000e+00 1.0000000e+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.]
```

```
A * A⁻¹ (Identity Matrix):
 [[1.0000000e+00 4.4408921e-16]
 [0.0000000e+00 1.0000000e+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.]
```