Worksheet 0:

Name: Nijal Shankar

Group: L6CG4

Uni ID: 2329790

Importing Necessary Libraries

```
[2] import numpy as np
    import time
```

```
[3] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Problem - 1: Array Creation

1. Initializing Empty Array (2 * 2)

```
[4] array = np.empty((2, 2))
    array #contains garbage values
```

```
array([[1.68016767e-313, 0.00000000e+000],
       [4.94065646e-324,             nan]])
```

```
[5] array = np.zeros((2, 2))
    array #initializes with 0
```

```
array([[0., 0.],
       [0., 0.]])
```

```
[6] array = np.full((2, 2), 20) #you can specify your initialization value
    array #20
```

```
array([[20, 20],
       [20, 20]])
```

2. Initializing all one array (4 * 2)

```
[7] array = np.ones((4, 2), dtype=int) #set type to integer initiall float
    array
```

```
array([[1, 1],
       [1, 1],
       [1, 1],
       [1, 1]])
```

```
array = np.full((4, 2), 1)
array
```

```
array([[1, 1],
       [1, 1],
       [1, 1],
       [1, 1]])
```

3. New Array of Given Shape and type filled with fill value

```
[9] array = np.full((4, 4), 9, dtype=int)
    array
```

```
array([[9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9]])
```

4. New array of zeros with same shape and type as given array

4. New array of zeros with same shape and type as given array

```
[10] array = np.full((4, 4), 8)
     new_array = np.zeros_like(array)
     new_array
```

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

5. New array of ones with same shape and type as given array

```
[11] array = np.full((4, 3), 8)
     new_array = np.ones_like(array)
     new_array
```

```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

6. Convert to NumPy array

```
[12] new_list = [1, 2, 3, 4]
     np_list = np.array(new_list)
     print("Normal List: ", new_list)
     print("Numpy List: ", np_list)
```

```
Normal List:  [1, 2, 3, 4]
Numpy List:  [1 2 3 4]
```

Problem - 2: Array Manipulation: Numerical Ranges and Array Indexing

Problem - 2: Array Manipulation: Numerical Ranges and Array Indexing

1. Create an array with values ranging from 10 to 49

```
[13] array = np.arange(10, 50)
     array
```

```
array([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])
```

2. Create a 3X3 matrix with values ranging from 0 to 8

```
[14] array = np.arange(0, 9)
     reshaped = np.reshape(array, (3, 3))
     print(array)
     print(reshaped)
```

```
[0 1 2 3 4 5 6 7 8]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

3. Create a 3*3 identity matrix

```
[15] array = np.eye(3, dtype=int)
     array
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

+ Code + Text

4. Random Array Size 30 and Mean

```
[16] random_arr = np.random.random(30)

     mean_arr = random_arr.mean()

     print(random_arr)
     print("Mean: ", mean_arr)
```

```
[0.48805522 0.3283804  0.5516419  0.27518501 0.72231566 0.46915415
 0.10101993 0.67113209 0.77322739 0.41294299 0.90668634 0.10264411
 0.482323   0.69149765 0.63790128 0.18062457 0.89415753 0.51565466
 0.74151858 0.41187509 0.45544561 0.99806441 0.01201314 0.57103502
 0.13955009 0.11300811 0.55188567 0.3902955  0.98572201 0.57677446]
Mean:  0.5050577185398658
```

5. Create a 10X10 array with random values and find the minimum and maximum values

```
[17] # random_arr = np.random.random((10, 10))
     random_arr = np.random.randint(1, 10, (10, 10))
     min = random_arr.min()
     max = random_arr.max()

     print(random_arr)
     print()
     print("Min: ", min)
     print("Max: ", max)
```

```
[[1 2 6 6 2 1 6 9 1 7]
 [2 2 1 8 6 8 1 6 9 1]
 [2 3 7 8 1 6 5 6 9 7]
 [7 6 7 9 4 1 9 8 6 3]
 [6 9 1 2 5 6 5 3 5 3]
 [7 2 3 3 5 2 2 7 7 3]
 [6 7 5 1 1 8 4 3 9 5]
 [9 9 5 4 8 9 8 4 1 4]
 [3 8 7 4 4 3 2 9 7 9]
 [5 6 5 9 8 9 7 4 9 3]]
```

```
Min:  1
Max:  9
```

6. Create a zero array of size 10 and replace 5th element with 1

```
[18] array = np.zeros(10, dtype=int)
     print(array)
     array[4] = 1
     print(array)
```

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

7. Reverse an array

```
[19] array = [1 , 0, 0, 3, 5, 1]
     rev_arr = array[::-1]
     rev_arr
```

```
[1, 5, 3, 0, 0, 1]
```

8. Create a 2d array with 1 on border and 0 inside

```
[20] arr = np.random.randint(1, 10, (7, 7))

     # arr = np.zeros_like(arr)
     arr[0, :] = 1
     arr[-1, :] = 1
     arr[:, 0] = 1
     arr[:, -1] = 1

     arr[0:-1, 1:-1] = 0

     arr
```

```
array([[1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 1, 1, 1, 1, 1, 1]])
```

9. Create a 8X8 matrix and fill it with a checkerboard pattern

```
[21] array = np.zeros((9, 9), dtype=int)

     array[1::2, 0::2] = 1
     array[::2, 1::2] = 1
     array
```

```
array([[0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0]])
```

Problem - 3: Array Operations

```
[22] x = np.array([[1, 2], [3, 5]])
     y = np.array([[5, 6], [7, 8]])
     v = np.array([9, 10])
     w = np.array([11, 12])
```

1. Add Arrays

1. Add Arrays

```
[23] sum = x + y
     sum1 = v + w
     print(sum)
     print()
     print(sum1)
```

```
[[ 6  8]
 [10 13]]

[20 22]
```

2. Subtract Arrays

```
[24] sub = x - y
     sub1 = v - w
     print(sub)
     print()
     print(sub1)
```

```
[[-4 -4]
 [-4 -3]]

[-2 -2]
```

3. Multiply Array With Integer

```
[25] mulArr = 7 * x
     mulArr
```

```
array([[ 7, 14],
       [21, 35]])
```

4. Square of Each Element of Array

4. Square of Each Element of Array

```
[26] powArr = x ** 2
     powArr
```

```
array([[ 1,  4],
       [ 9, 25]])
```

5. Dot Product

```
[27] vDotw = np.dot(v, w)
     xDotv = np.dot(x, v)
     xDoty = np.dot(x, y)

     print(f"V.W: {vDotw}")
     print(f"X.V: {xDotv}")
     print(f"X.Y: \n{xDoty}")
```

```
V.W: 219
X.V: [29 77]
X.Y:
[[19 22]
 [50 58]]
```

6. Concatenate - 1

```
[28] conxy = np.concatenate((x, y), axis = 0)
     convw = np.vstack((v, w))
     print(conxy)
     print()
     print(convw)
```

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

[[ 9 10]
```

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

7. Concatenate - 2 (Dimension Mismatch)

```
[29] conxv = np.concatenate((x, v), axis = 0)
     conxv
     ##This cause error because the arrays should have the same number of dimensions
     #x is a 2D array where as v is a 1D array
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-ee772db1a997> in <cell line: 0>()
----> 1 conxv = np.concatenate((x, v), axis = 0)
      2 conxv
      3 ##This cause error because the arrays should have the same number of dimensions
      4 #x is a 2D array where as v is a 1D array

ValueError: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array a
```

Next steps: Explain error

Problem - 4: Matrix Operations

```
[30] A = np.array([[3, 4], [7, 8]])
     B = np.array([[5, 3], [2, 1]])
```

1. A.A^-1 = I

```
[31] A_Inverse = np.linalg.inv(A)
     proof = np.round(np.matmul(A, A_Inverse))
     proof
```

```
array([[1., 0.],
       [0., 1.]])
```

2. AB != BA

```
[32] AB = np.matmul(A, B)
     BA = np.matmul(B, A)

     print(f"AB:\n{AB} \nBA:\n{BA}")
```

```
AB:
[[23 13]
 [51 29]]
BA:
[[36 44]
 [13 16]]
```

3. (AB)T = BT.AT

```
[33] AB = np.matmul(A, B)
     AB_T = AB.T
     B_T = B.T
     A_T = A.T
     B_T_Dot_A_T = np.matmul(B_T, A_T)

     print(f"AB_T: \n{AB_T} \n\n B_T.A_T: \n{B_T_Dot_A_T}")
```

```
AB_T:
[[23 51]
 [13 29]]

 B_T.A_T:
[[23 51]
 [13 29]]
```

Linear Equation Using Inverse Method

Linear Equation Using Inverse Method

```
[34] A = np.array([[2, -3, 1],
                   [1, -1, 2],
                   [3, 1,  -1]])
     B = np.array([-1, -3, 9])

     A_Inverse = np.linalg.inv(A)

     X = np.matmul(A_Inverse, B)

     print(f"[x y z] = {X}")
```

```
[x y z] = [ 2.  1. -2.]
```

Numpy Speed

Numpy Speed

```
[35] import numpy as np
     import time
     import scipy.sparse as sparse
     import builtins

     size = 1000000
     matrices = 1000

     # Create Python lists and NumPy arrays
     pyArr1 = list(range(size))
     pyArr2 = list(range(size))
     npArr1 = np.arange(size)
     npArr2 = np.arange(size)

     # Time addition
     sTime = time.time()
     pySum = [pyArr1[i] + pyArr2[i] for i in range(size)]
     pyTimeAdd = time.time() - sTime

     sTime = time.time()
     npSum = npArr1 + npArr2
     npTimeAdd = time.time() - sTime

     print(f"Addition Time: \nNumpy: {npTimeAdd:.5f} \nNormal Py list: {pyTimeAdd:.5f}\n")

     # Time element-wise multiplication
     sTime = time.time()
     pyMul = [pyArr1[i] * pyArr2[i] for i in range(size)]
     pyTimeMul = time.time() - sTime

     sTime = time.time()
     npMul = npArr1 * npArr2
     npTimeMul = time.time() - sTime

     print(f"Element Multiplication Time: \nNumpy: {npTimeMul:.5f} \nNormal Py list: {pyTimeMul:.5f}\n")

     # Time dot product
     sTime = time.time()
     pyDot = builtins.sum(pyArr1[i] * pyArr2[i] for i in range(size))  # Use builtins.sum to avoid issues
     pyTimeDot = time.time() - sTime

     sTime = time.time()
     npDot = np.dot(npArr1, npArr2)
     npTimeDot = time.time() - sTime

     print(f"Dot Product Time: \nNumpy: {npTimeDot:.5f} \nNormal Py list: {pyTimeDot:.5f}\n")
```

```python
# Create Python and NumPy matrices
pyMat1 = [[j for j in range(matrices)] for i in range(matrices)]
pyMat2 = [[j for j in range(matrices)] for i in range(matrices)]

# Use sparse matrices to save memory
npMat1 = sparse.csr_matrix(np.arange(matrices**2).reshape(matrices, matrices))
npMat2 = sparse.csr_matrix(np.arange(matrices**2).reshape(matrices, matrices))

# Time matrix multiplication
sTime = time.time()
pyMatMul = [[builtins.sum(pyMat1[i][k] * pyMat2[k][j] for k in range(matrices)) for j in range(matrices)] for i in range(matrices)]
pyTimeMatMul = time.time() - sTime

sTime = time.time()
npMatMul = npMat1 @ npMat2  # Use sparse matrix multiplication
npTimeMatMul = time.time() - sTime

print(f"Matrix Multiplication Time: \nNumpy: {npTimeMatMul:.5f} \nNormal Py list: {pyTimeMatMul:.5f}\n")
```

```
Addition Time:
Numpy: 0.01720
Normal Py list: 0.20257

Element Multiplication Time:
Numpy: 0.00678
Normal Py list: 0.21229

Dot Product Time:
Numpy: 0.00303
Normal Py list: 0.19824

Matrix Multiplication Time:
Numpy: 3.77690
Normal Py list: 188.36880
```

4.1 Exercise on Functions:

4.1 Exercise on Functions:

Task - 1:

[36]
```
def convert(value, from_unit, to_unit):
    """
    Generic converter for length, weight, and volume.

    Parameters:
    value (float): The numeric value to convert.
    from_unit (str): The unit to convert from.
    to_unit (str): The unit to convert to.

    Returns:
    float: The converted value.
    """
    conversions = {
        ('m', 'ft'): 3.28084,
        ('ft', 'm'): 1 / 3.28084,
        ('kg', 'lbs'): 2.20462,
        ('lbs', 'kg'): 1 / 2.20462,
        ('l', 'gal'): 0.264172,
        ('gal', 'l'): 1 / 0.264172
    }

    if (from_unit, to_unit) in conversions:
        return value * conversions[(from_unit, to_unit)]
    else:
        raise ValueError("Unsupported conversion.")


def main():
    print("Unit Conversion Program")
    print("1. Length (meters <-> feet)")
    print("2. Weight (kilograms <-> pounds)")
    print("3. Volume (liters <-> gallons)")

    try:
        choice = input("Enter your choice (1/2/3): ")

        units = {
            '1': ('m', 'ft'),
            '2': ('kg', 'lbs'),
            '3': ('l', 'gal')
        }

        if choice not in units:
            print("Invalid choice. Please enter 1, 2, or 3.")
            return
```

```
            from_unit = input(f"Convert from {units[choice][0]} or {units[choice][1]}: ").strip().lower()
            if from_unit not in units[choice]:
                print("Invalid unit.")
                return

            to_unit = units[choice][1] if from_unit == units[choice][0] else units[choice][0]

            value = float(input(f"Enter the value in {from_unit}: "))

            result = convert(value, from_unit, to_unit)
            print(f"{value} {from_unit} is equal to {result:.2f} {to_unit}")

        except ValueError:
            print("Please enter a valid number.")
        except Exception as e:
            print(f"Unexpected error: {e}")


    if __name__ == "__main__":
        main()
```

```
Unit Conversion Program
1. Length (meters <-> feet)
2. Weight (kilograms <-> pounds)
3. Volume (liters <-> gallons)
Enter your choice (1/2/3): 2
Convert from kg or lbs: 60
Invalid unit.
```

Task2

Task2

```python
import numpy as np

def find_sum(numbers):
    """
    Calculate the sum of a list of numbers.

    Parameters:
    numbers (list): A list of numbers.

    Returns:
    float: The sum of the numbers.
    """
    return np.sum(numbers)  # Use numpy sum

def find_average(numbers):
    """
    Calculate the average of a list of numbers.

    Parameters:
    numbers (list): A list of numbers.

    Returns:
    float: The average of the numbers.
    """
    return np.mean(numbers)  # Use numpy mean

def find_maximum(numbers):
    """
    Find the maximum value in a list of numbers.

    Parameters:
    numbers (list): A list of numbers.

    Returns:
    float: The maximum value.
    """
    return np.max(numbers)  # Use numpy max

def find_minimum(numbers):
    """
    Find the minimum value in a list of numbers.

    Parameters:
    numbers (list): A list of numbers.

    Returns:
    float: The minimum value.
    """
    return np.min(numbers)  # Use numpy min
```

```python
def main():
    print("Mathematical Operations on a List of Numbers")
    print("1. Find Sum")
    print("2. Find Average")
    print("3. Find Maximum")
    print("4. Find Minimum")

    try:
        # Get the operation choice
        choice = input("Choose an operation (1/2/3/4): ")

        # Check if the choice is valid
        if choice not in ['1', '2', '3', '4']:
            print("Invalid choice. Please choose 1, 2, 3, or 4.")
            return

        # Get the list of numbers
        numbers_input = input("Enter a list of numbers separated by spaces: ").strip()

        if not numbers_input:
            raise ValueError("No numbers entered. Please provide a valid list of numbers.")

        # Convert the input to a list of floats
        numbers = [float(num) for num in numbers_input.split()]

        if len(numbers) == 0:
            raise ValueError("The list is empty. Please enter at least one number.")

        print(f"Operation: {choice}")
        print(f"Numbers entered: {numbers}")

        # Perform the chosen operation
        if choice == '1':
            result = find_sum(numbers)
            print(f"The sum of the numbers is: {result}")
        elif choice == '2':
            result = find_average(numbers)
            print(f"The average of the numbers is: {result}")
        elif choice == '3':
            result = find_maximum(numbers)
            print(f"The maximum value is: {result}")
        elif choice == '4':
            result = find_minimum(numbers)
            print(f"The minimum value is: {result}")

    except ValueError as ve:
        print(f"Error: {ve}")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

```python
if __name__ == "__main__":
    main()
```

```
Mathematical Operations on a List of Numbers
1. Find Sum
2. Find Average
3. Find Maximum
4. Find Minimum
Choose an operation (1/2/3/4): 2
Enter a list of numbers separated by spaces: 10 2 9 21
Operation: 2
Numbers entered: [10.0, 2.0, 9.0, 21.0]
The average of the numbers is: 10.5
```

4.2 Exercise on List Manipulation:

4.2 Exercise on List Manipulation:

1. Extract Every Other Element:

```
[54] #extract every other element

     def extract_every_other(lst):
         """
         Extract every other element from the list, starting from the first element.

         Parameters:
         lst (list): The list from which to extract every other element.

         Returns:
         list: A new list containing every other element from the original list.
         """
         return lst[::2]  # Using list slicing to extract every other element

     # Taking input from the user
     user_input = input("Enter a list of numbers separated by spaces: ")
     lst = [int(num) for num in user_input.split()]

     # Calling the function and displaying the result
     result = extract_every_other(lst)
     print("Every other element:", result)
```

```
Enter a list of numbers separated by spaces: 20 9 10 2
Every other element: [20, 10]
```

2. Slice a Sublist:

```
[39] #2. Slice a Sublist:

     def get_sublist(lst, start, end):
         """
         Returns a sublist from the given list, starting from the 'start' index and ending at the 'end' index (inclusive).

         Parameters:
         lst (list): The list to slice.
         start (int): The starting index (inclusive).
         end (int): The ending index (inclusive).
```

```
        Returns:
        list: A sublist from the 'start' to 'end' indices.
        """
        return lst[start:end+1]  # Slicing from start to end (inclusive)

    # Example usage:
    lst = [1, 2, 3, 4, 5, 6]
    start = input("Enter the start index: ")
    end = input("Enter the end index: ")

    # Convert start and end to integers
    start = int(start)
    end = int(end)
    result = get_sublist(lst, start, end)
    print(result)
```

```
Enter the start index: 2
Enter the end index: 9
[3, 4, 5, 6]
```

3. Reverse a List Using Slicing:

```
[40] # Reverse a List Using Slicing
     def reverse_list(lst):
         """
         Reverses the given list using slicing.

         Parameters:
         lst (list): The list to reverse.

         Returns:
         list: The reversed list.
         """
         return lst[::-1]  # Using slicing to reverse the list


     lst = [1, 2, 3, 4, 5]
     result = reverse_list(lst)
     print(result)
```

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

4. Remove the First and Last Elements:

4. Remove the First and Last Elements:

```
[41] #4. Remove the First and Last Elements:
     def remove_first_last(lst):
         """
         Removes the first and last elements of the list and returns the resulting sublist.

         Parameters:
         lst (list): The list from which to remove the first and last elements.

         Returns:
         list: The sublist without the first and last elements.
         """
         return lst[1:-1]  # Slicing from the second element to the second-to-last element


     lst = [1, 2, 3, 4, 5]
     print(remove_first_last(lst))
```

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

5.Extract Elements from the End

```
[53] #5.Extract Elements from the End
     def get_first_n(lst, n):
         """
         Extracts the first n elements from the list.
         """
         return lst[:n]  # Slicing the first n elements

     # Taking input from user
     lst_input = input("Enter the list of numbers separated by spaces: ")
     lst = [int(x) for x in lst_input.split()]
     n = int(input("Enter the number of elements to extract from the start: "))
     print(get_first_n(lst, n))
```

```
Enter the list of numbers separated by spaces: 2 4 1
Enter the number of elements to extract from the start: 1
[2]
```

6. Extract Elements from the End:

```
[43] #6. Extract Elements from the End:
     def get_last_n(lst, n):
         """
         Extracts the last n elements from the list.
         """
```

```
    """
    return lst[-n:]  # Slicing the last n elements

# Taking input from user
lst_input = input("Enter the list of numbers separated by spaces: ")
lst = [int(x) for x in lst_input.split()]
n = int(input("Enter the number of elements to extract from the end: "))
print(get_last_n(lst, n))
```

```
Enter the list of numbers separated by spaces: 2 4 5
Enter the number of elements to extract from the end: 2
[4, 5]
```

7.Extract Elements in Reverse Order

```
[44] #7. Extract Elements in reverse order:
def reverse_skip(lst):
    """
    Extracts elements in reverse order starting from the second-to-last element,
    skipping one element in between.
    """
    return lst[-2::-2]  # Slicing to get every second element in reverse order starting from the second-to-last

# Taking input from user
lst_input = input("Enter the list of numbers separated by spaces: ")
lst = [int(x) for x in lst_input.split()]
print(reverse_skip(lst))
```

```
Enter the list of numbers separated by spaces: 9 20 39 8
[39, 9]
```

4.3 Exercise on Nested List:

1.Flatten a Nested List:

```
[45] # 1. Flatten a Nested List:
def flatten(lst):
    """
    Flattens a nested list into a single list.

    Parameters:
    lst (list): The nested list to flatten.

    Returns:
    list: A flattened list containing all elements.
    """
```

```
        """
        flattened = []
        for item in lst:
            if isinstance(item, list):
                flattened.extend(flatten(item))  # Recursively flatten if the item is a list
            else:
                flattened.append(item)
        return flattened


    nested_lst = [[1, 2], [3, 4], [5]]
    flattened_lst = flatten(nested_lst)
    print("Flattened List:", flattened_lst)
```

```
Flattened List: [1, 2, 3, 4, 5]
```

2. Accessing Nested List Elements:

```
[46] # 2. Accessing Nested List Elements:
    def access_nested_element(lst, indices):
        """
        Extracts an element from a nested list based on a list of indices.

        Parameters:
        lst (list): The nested list to access.
        indices (list): A list of indices to access the element.

        Returns:
        element: The element at the specified indices.
        """
        for index in indices:
            lst = lst[index]
        return lst


    nested_lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    indices = [1, 1]
    element = access_nested_element(nested_lst, indices)
    print("Accessed Element:", element)
```

```
Accessed Element: 5
```

3. Sum of All Elements in a Nested List:

```
[47] # 3. Sum of All Elements in a Nested List:
    def sum_nested(lst):
        """
        Calculates the sum of all elements in a nested list, regardless of depth.
```

```
        Parameters:
        lst (list): The nested list to sum.

        Returns:
        int: The sum of all elements.
        """
        total = 0
        for item in lst:
            if isinstance(item, list):
                total += sum_nested(item)  # Recursively sum if the item is a list
            else:
                total += item
        return total


    nested_lst = [[6, 2], [9, [4, 5]], 6]
    sum_result = sum_nested(nested_lst)
    print("Sum of All Elements:", sum_result)
```

```
Sum of All Elements: 32
```

Double-click (or enter) to edit

```
[48] # 4. Remove Specific Element from a Nested List:
    def remove_element(lst, elem):
        """
        Removes all occurrences of a specific element from a nested list.

        Parameters:
        lst (list): The nested list to modify.
        elem: The element to remove.

        Returns:
        list: The modified list with the element removed.
        """
        for i in range(len(lst)):
            if isinstance(lst[i], list):
                lst[i] = remove_element(lst[i], elem)
            else:
                if lst[i] == elem:
                    lst[i] = None  # Set to None to delete
        return [item for item in lst if item is not None]

    # Example usage:
    nested_lst = [[1, 2], [3, 2], [4, 5]]
    print("Original List:", nested_lst)
    elem_to_remove = int(input("Enter the element to remove: "))
    removed_lst = remove_element(nested_lst, elem_to_remove)
    print("List After Removal:", removed_lst)
```

```
    print("List After Removal:", removed_lst)
```

```
Original List: [[1, 2], [3, 2], [4, 5]]
Enter the element to remove: 2
List After Removal: [[1], [3], [4, 5]]
```

5. Find the Maximum Element in a Nested List:

```
[49] import builtins

    # 5. Find the Maximum Element in a Nested List:
    def find_max(lst):
        """
        Finds the maximum element in a nested list, regardless of depth.

        Parameters:
        lst (list): The nested list to find the maximum value in.

        Returns:
        int: The maximum value.
        """
        max_val = float('-inf')
        for item in lst:
            if isinstance(item, list):
                max_val = builtins.max(max_val, find_max(item))  # Use the built-in max explicitly
            else:
                max_val = builtins.max(max_val, item)  # Ensure built-in max is used
        return max_val

    # Example usage:
    nested_lst = [[1, 2], [3, [4, 5]], 6]
    max_element = find_max(nested_lst)
    print("Maximum Element:", max_element)
```

```
Maximum Element: 6
```

6. Count Occurrences of an Element in a Nested List:

```
[50] # 6. Count Occurrences of an Element in a Nested List:
    def count_occurrences(lst, elem):
        """
        Counts how many times a specific element appears in a nested list.

        Parameters:
        lst (list): The nested list to search.
        elem: The element to count.
```

```
        Returns:
        int: The count of the element in the list.
        """
        count = 0
        for item in lst:
            if isinstance(item, list):
                count += count_occurrences(item, elem)  # Recursively count if the item is a list
            else:
                if item == elem:
                    count += 1
        return count

    # Example usage:
    nested_lst = [[1, 2], [2, 3], [2, 4]]
    elem_to_count = 2
    occurrences = count_occurrences(nested_lst, elem_to_count)
    print("Occurrences of Element:", occurrences)
```

```
Occurrences of Element: 3
```

7. Flatten a List of Lists of Lists:

```
[51] # 7. Flatten a List of Lists of Lists:
    def deep_flatten(lst):
        """
        Flattens a deeply nested list of lists into a single list.

        Parameters:
        lst (list): The deeply nested list to flatten.

        Returns:
        list: A flattened list containing all elements.
        """
        flattened = []
        for item in lst:
            if isinstance(item, list):
                flattened.extend(deep_flatten(item))  # Recursively flatten if the item is a list
            else:
                flattened.append(item)
        return flattened

    deep_nested_lst = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
    deep_flattened_lst = deep_flatten(deep_nested_lst)
    print("Deep Flattened List:", deep_flattened_lst)
```

```
Deep Flattened List: [1, 2, 3, 4, 5, 6, 7, 8]
```

8. Nested List Average:

```
[52] # 8. Nested List Average:
     def average_nested(lst):
         """
         Calculates the average of all elements in a nested list.

         Parameters:
         lst (list): The nested list to calculate the average from.

         Returns:
         float: The average of all elements.
         """
         total = 0
         count = 0
         for item in lst:
             if isinstance(item, list):
                 sub_total, sub_count = average_nested(item)
                 total += sub_total
                 count += sub_count
             else:
                 total += item
                 count += 1
         return total, count


     def get_average(lst):
         total, count = average_nested(lst)
         return total / count if count != 0 else 0

     # Example usage:
     nested_lst = [[1, 2], [3, 4], [5, 6]]
     avg_result = get_average(nested_lst)
     print("Average of Elements:", avg_result)
```

```
Average of Elements: 3.5
```