```
main()
```

```
Enter the numbers separated by space: 1 3 6 8 5
Choose 1 for sum, 2 for average, 3 for maximum, 4 for minimum: 3
Maximum of the numbers: 8.0
```

```
Unit conversion
Choose 1 for length, 2 for weight, 3 for volume: 1
Enter the value: 2
Enter the unit to convert from (m/ft): m
6.56168
```

Extract Every Other Element:

[ ] ↳ 1 cell hidden

Slice a Sublist:

```
def get_sublist(lst, start, end):
  return lst[start:end]

new_lst = get_sublist([1, 2, 3, 4, 5, 6],2, 4)
new_lst
```

```
[3, 4]
```

```
[6] def reverse_list(lst):
      return lst[::-1]
    reverse_list([1, 2, 3, 4, 5])
```

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

Remove the First and Last Elements:

```
[7] def remove_first_last(lst):
      return lst[1:-1:]

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

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

Get the First n Elements:

```
[8] def get_first_n(lst, n):
      return lst[:n]
    n = int(input("Enter n number to return first n element"))
    get_first_n([1, 2, 3, 4, 5],n)
```

```
Enter n number to return first n element4
[1, 2, 3, 4]
```

Extract Elements from the End:

```
def get_lst_n(lst, n):
  return lst[-n:]
n = int(input("Enter number to return n last element"))

get_lst_n([1, 2, 3, 4, 5],n)
```

```
Enter number to return n last element5
[1, 2, 3, 4, 5]
```

Extract Elements in Reverse Order

```
[10] def reverse_skip(lst):
       return lst[-2::-2]

     reverse_skip([1, 2, 3, 4, 5, 6])
```

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

Exercise on Nested List:

Flatten a Nested List:

```
[11] def flatten(lst):

       flat_list = []

       for sublist in lst:
         if isinstance(sublist, list):
           flat_list.extend(sublist)
         else:
           flat_list.append(sublist)
       return flat_list

     nested_list = [[1, 2], [3, 4], [5]]
     print(flatten(nested_list))
```

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

```
[12] def access_nested_element(lst, indices):
       return lst[indices[0]][indices[1]]

     access_nested_element( [[1, 2, 3], [4, 5, 6], [7, 8, 9]], [1,2])
```

```
6
```

```
[13] def sum_nested(lst):
       total = 0
       for item in lst:
         if isinstance(item, list):
             total += sum_nested(item)
         else:
             total += item
       return total


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

```
21
```

[14]
```
def remove_element(lst, elm):
  for i, sublist in enumerate(lst):
    for j, num in enumerate(sublist):
      if num == 2:
        lst[i].pop(j)

  return lst

remove_element([[1, 2], [3, 2], [4, 5]], 2)
```

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

[15]
```
def find_max(lst):
    max_value = float('-inf')  # Initialize with negative infinity

    for item in lst:
        if isinstance(item, list):
            max_value = max(max_value, find_max(item))  # Recursively find max
        else:
            max_value = max(max_value, item)  # Compare numbers

    return max_value


nested_list = [[1, 2], [3, [4, 5]], 6]
find_max(nested_list)
```

```
6
```

[16]
```
def count_cccurrences(lst, elem):
  count = 0
  for item in lst:
    if isinstance(item,list):
      for num in item:
        if elem == num:
          count +=1
    else:
      if elem == item:
        count +=1

  return count

lst =  [[1, 2], [2, 3], [2, 4]]
count_cccurrences(lst,2)
```

```
3
```

[17]
```python
def deep_flatten(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):  # If item is a list, recursively flatten it
            flat_list.extend(deep_flatten(item))
        else:
            flat_list.append(item)  # Append non-list elements directly
    return flat_list


nested_list = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
deep_flatten(nested_list)
```

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

[18]
```python
def deep_flatten(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):  # If item is a list, recursively flatten it
            flat_list.extend(deep_flatten(item))
        else:
            flat_list.append(item)  # Append non-list elements directly
    return flat_list


nested_list =[[1, 2], [3, 4], [5, 6]]
new_list = deep_flatten(nested_list)

sum(new_list)/len(new_list)
```

```
3.5
```

Problem - 1: Array Creation:

```
[19] import numpy as np
```

Initialize an empty array with size 2X2

```
[20] empty_array = np.empty((2,2))
     empty_array
```

```
array([[6.74343392e-310, 3.03215439e-316],
       [1.93673733e-320, 0.00000000e+000]])
```

Initialize an all-one array with size 4X2

```
[21] one_array = np.ones((4,2))
     print(one_array)
```

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

Return a new array of given shape and type, filled with fill value

```
[22] fill_value_array = np.full((3,2), 7) #create 3X2 size of matrix with value 7 on each
     fill_value_array
```

```
array([[7, 7],
       [7, 7],
       [7, 7]])
```

Return a new array of zeros with same shape and type as a given array

```
[23] sample_array = np.array([[4, 5], [6, 7]])
     print(sample_array)
     print("\nZero Array:")
     zero_array = np.zeros_like(sample_array)
     print(zero_array)
```

```
[[4 5]
 [6 7]]

Zero Array:
[[0 0]
 [0 0]]
```

Return a new array of ones with same shape and type as a given array

```
[24] ones_like_array = np.ones_like(sample_array)
     ones_like_array
```

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

Convert an existing list to a NumPy array

```
[25] new_list = [1, 2, 3, 4]
     numpy_array = np.array(new_list)
     numpy_array
```

```
array([1, 2, 3, 4])
```

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

Create an array with values ranging from 10 to 49

```
[26] array_10_49 = np.arange(10, 50)
     array_10_49
```

```
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])
```

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

```
[27] matrix_3x3 = np.arange(9).reshape(3, 3)
     matrix_3x3
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Create a 3X3 identity matrix.{Hint:np.eye()} **bold text**

```
[28] identity_matrix = np.eye(3)
     identity_matrix
```

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

Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function} **bold text**

```
[29] random_array = np.random.random(30)
     random_array.mean()
```

```
0.5784220300454481
```

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

```
[30] random_matrix  = np.random.random((10,10))
     # print(random_matrix)
     print(f"Minimum value: {random_matrix.min()}")
     print(f"Maximum value: {random_matrix.max()}")
```

```
Minimum value: 0.0007818612852977802
Maximum value: 0.9746247645328001
```

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

```
[31] zero_array  = np.zeros(10)
     zero_array[4] = 1
     zero_array
```

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

Reverse an array arr = [1,2,0,0,4,0].

```
[32] arr = np.array([1,2,0,0,4,0])
     reversed_arr = arr[::-1]
     reversed_arr
```

```
array([0, 4, 0, 0, 2, 1])
```

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

```
[33] boarder_array = np.ones((5,5))
     boarder_array[1:-1, 1:-1] = 0
     boarder_array
```

```
array([[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.]])
```

```
array([[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]])
```

Problem - 3: Array Operations:

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

```
[36] x+y
```

```
array([[ 6,  8],
       [10, 13]])
```

```
[37] x-y
```

```
array([[-4, -4],
       [-4, -3]])
```

```
[38] x*3
```

```
array([[ 3,  6],
       [ 9, 15]])
```

```
[39] np.square(x)
```

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

```
[40] dot_vw = np.dot(v, w)  # Dot product of v and w
     dot_xv = np.dot(x, v)  # Dot product of x and v
     dot_xy = np.dot(x, y)  # Dot product of x and y

     print("\nDot product of v and w:", dot_vw)
     print("\nDot product of x and v:\n", dot_xv)
     print("\nDot product of x and y:\n", dot_xy)
```

```

Dot product of v and w: 219

Dot product of x and v:
 [29 77]

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

```
Concatenation of x and y along rows:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]

Concatenation of v and w along columns:
 [[ 9 10]
 [11 12]]
```

1. Prove $A \cdot A$ - 1 = I A·A −1 =I

```
[44] A = np.array([[3, 4], [7, 8]])

     # Compute the inverse of A
     A_inv = np.linalg.inv(A)

     # Multiply A by its inverse
     I = np.dot(A, A_inv)

     # Print results
     print("A * A^-1:\n", I)

     # Check if it's an identity matrix
     print("\nIs A * A^-1 approximately equal to I? ", np.allclose(I, np.eye(2)))
```

```
A * A^-1:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]

Is A * A^-1 approximately equal to I?  True
```

Solving the Linear System Using the Inverse Method

```
[47] # Define matrix A (coefficients)
     A = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])

     # Define matrix B (constants)
     B = np.array([-1, -3, 9])

     # Solve for X using inverse
     A_inv = np.linalg.inv(A)  # Compute inverse of A
     X = np.dot(A_inv, B)  # Compute X

     # Print results
     print("\nSolution for x, y, z:\n", X)
```

```
Solution for x, y, z:
 [ 2.  1. -2.]
```

Solving Using np.linalg.solve

```
[48] # Solve directly using np.linalg.solve
     X_solve = np.linalg.solve(A, B)

     # Print results
     print("\nSolution using np.linalg.solve:\n", X_solve)
```

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