

**Student ID = 2330545**

**Name = Renjen Sherpa**

TO - DO - Task

Create a Python program that converts between different units of measurement. • The program should:

1. Prompt the user to choose the type of conversion (e.g., length, weight, volume).
2. Ask the user to input the value to be converted.
3. Perform the conversion and display the result.
4. Handle potential errors, such as invalid input or unsupported conversion types.

• Requirements:

1. Functions: Define at least one function to perform the conversion.
2. Error Handling: Use try-except blocks to handle invalid input (e.g., non-numeric values).
3. User Input: Prompt the user to select the conversion type and input the value.
4. Docstrings: Include a docstring in your function to describe its purpose, parameters, and return value.

• Conversion Options:

1. Length:

– Convert meters (m) to feet (ft). – Convert feet (ft) to meters (m).

## 2. Weight:

- Convert kilograms (kg) to pounds (lbs). – Convert pounds (lbs) to kilograms (kg).

## 3. Volume:

- Convert liters (L) to gallons (gal). – Convert gallons (gal) to liters (L).

✓  
20s

```
def convert_length(value, unit):  
    if unit == 'm':  
        return value * 3.28084  
    elif unit == 'ft':  
        return value / 3.28084  
    else:  
        return None  
  
def convert_weight(value, unit):  
    if unit == 'kg':  
        return value * 2.20462  
    elif unit == 'lbs':  
        return value / 2.20462  
    else:  
        return None  
  
def convert_volume(value, unit):  
    if unit == 'L':  
        return value * 0.264172
```



```
def convert_volume(value, unit):  
    if unit == 'L':  
        return value * 0.264172  
    elif unit == 'gal':  
        return value / 0.264172  
    else:  
        return None  
  
def main():  
    print("Unit Converter: Length, Weight, Volume")  
    print("1. Length (m <-> ft)")  
    print("2. Weight (kg <-> lbs)")  
    print("3. Volume (L <-> gal)")  
  
    try:  
        choice = int(input("Choose a conversion type (1-3): "))  
        value = float(input("Enter the value to convert: "))  
  
        if choice == 1:  
            unit = input("Enter unit (m or ft): ").lower()  
            result = convert_length(value, unit)  
            new_unit = 'ft' if unit == 'm' else 'm'  
        elif choice == 2:  
            unit = input("Enter unit (kg or lbs): ").lower()  
            result = convert_weight(value, unit)  
            new_unit = 'lbs' if unit == 'kg' else 'kg'
```

```

elif choice == 2:
    unit = input("Enter unit (kg or lbs): ").lower()
    result = convert_weight(value, unit)
    new_unit = 'lbs' if unit == 'kg' else 'kg'
elif choice == 3:
    unit = input("Enter unit (L or gal): ").lower()
    result = convert_volume(value, unit)
    new_unit = 'gal' if unit == 'L' else 'L'
else:
    print("Invalid choice. Please choose a valid option.")
    return

if result is not None:
    print(f"Converted value: {result:.2f} {new_unit}")
else:
    print("Invalid unit entered.")

except ValueError:
    print("Invalid input! Please enter a numerical value.")

if __name__ == "__main__":
    main()

```

2330545\_RenjenSherpa\_Worksheet - 0.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands | + Code + Text

```
print("Invalid input! Please enter a numerical value.")

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

Unit Converter: Length, Weight, Volume  
1. Length (m <-> ft)  
2. Weight (kg <-> lbs)  
3. Volume (L <-> gal)  
Choose a conversion type (1-3): 1  
Enter the value to convert: 100  
Enter unit (m or ft): ft  
Converted value: 30.48 m

2330545\_RenjenSherpa\_Worksheet - 0.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands | + Code + Text

```
def calculate_sum(numbers):
    return sum(numbers)

def calculate_average(numbers):
    return sum(numbers) / len(numbers) if numbers else 0

def find_maximum(numbers):
    return max(numbers)

def find_minimum(numbers):
    return min(numbers)

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

    try:
        choice = int(input("Choose an operation (1-4): "))
        numbers = list(map(float, input("Enter numbers separated by spaces: ").split()))

        if not numbers:
            print("Error: The list is empty.")
            return
```

```

▶ try:
    choice = int(input("Choose an operation (1-4): "))
    numbers = list(map(float, input("Enter numbers separated by spaces: ").split()))

    if not numbers:
        print("Error: The list is empty.")
        return

    if choice == 1:
        result = calculate_sum(numbers)
        print(f"Sum: {result}")
    elif choice == 2:
        result = calculate_average(numbers)
        print(f"Average: {result}")
    elif choice == 3:
        result = find_maximum(numbers)
        print(f"Maximum: {result}")
    elif choice == 4:
        result = find_minimum(numbers)
        print(f"Minimum: {result}")
    else:
        print("Invalid choice. Please choose a valid option.")

except ValueError:
    print("Invalid input! Please enter numbers correctly.")

```

```

    except ValueError:
        print("Invalid input! Please enter numbers correctly.")

if __name__ == "__main__":
    main()

```



Mathematical Operations on a List

1. Sum
2. Average
3. Maximum
4. Minimum

Choose an operation (1-4): 2

Enter numbers separated by spaces: 1 5 8 15

Average: 7.25

## 4.2 Exercise on List Manipulation:

1. Extract Every Other Element: Write a Python function that extracts every other element from a list, starting from the first element. • Requirements: – Define a function `extract_every_other(lst)` that takes a list `lst` as input and returns a new list containing every other element from the original list. – Example: For the input `[1, 2, 3, 4, 5, 6]`, the output should be `[1, 3, 5]`.

```
def extract_every_other(lst):  
    return lst[::2]  
  
input_list = [1, 2, 3, 4, 5, 6]  
result = extract_every_other(input_list)  
print(result) # Output: [1, 3, 5]
```

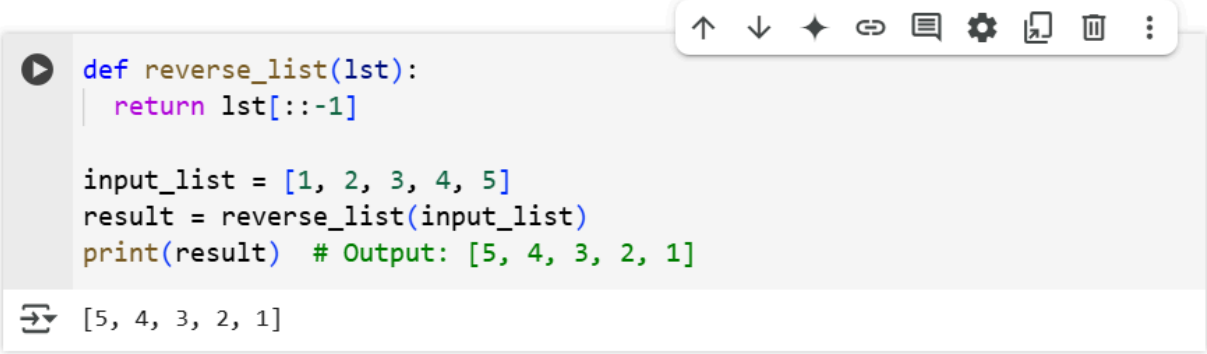
[1, 3, 5]

2. Slice a Sublist: Write a Python function that returns a sublist from a given list, starting from a specified index and ending at another specified index. • Requirements: – Define a function `get_sublist(lst, start, end)` that takes a list `lst`, a starting index `start`, and an ending index `end` as input and returns the sublist from `start` to `end` (inclusive). – Example: For the input `[1, 2, 3, 4, 5, 6]` with `start=2` and `end=4`, the output should be `[3, 4, 5]`.

```
def get_sublist(lst, start, end):  
    return lst[start:end+1]  
  
input_list = [1, 2, 3, 4, 5, 6]  
start_index = 2  
end_index = 4  
result = get_sublist(input_list, start_index, end_index)  
print(result) # Output: [3, 4, 5]
```

[3, 4, 5]

3. Reverse a List Using Slicing: Write a Python function that reverses a list using slicing. • Requirements: – Define a function reverse\_list(lst) that takes a list lst and returns a reversed list using slicing. – Example: For the input [1, 2, 3, 4, 5], the output should be [5, 4, 3, 2, 1].



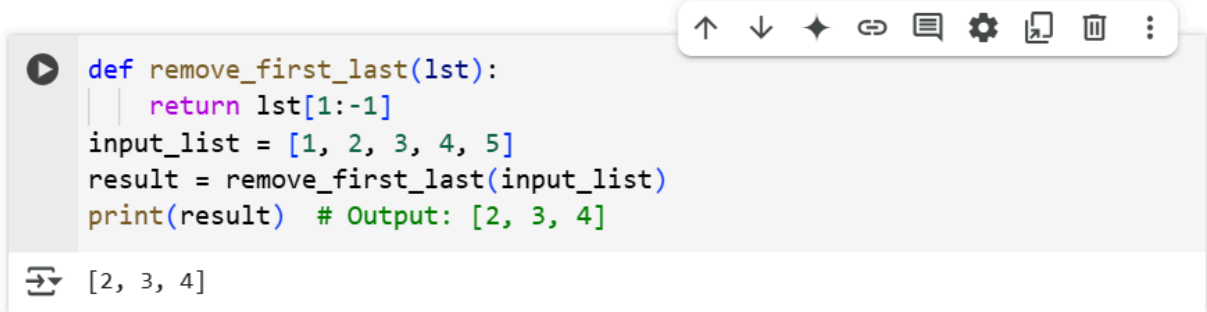
The image shows a Python IDE interface. At the top, there is a toolbar with icons for undo, redo, search, run, save, and other standard IDE functions. Below the toolbar, the code editor contains the following Python code:

```
def reverse_list(lst):  
    return lst[::-1]  
  
input_list = [1, 2, 3, 4, 5]  
result = reverse_list(input_list)  
print(result) # Output: [5, 4, 3, 2, 1]
```

Below the code editor, the output console shows the result of the execution:

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

4. Remove the First and Last Elements: Write a Python function that removes the first and last elements of a list and returns the resulting sublist. • Requirements: – Define a function remove\_first\_last(lst) that takes a list lst and returns a sublist without the first and last elements using slicing. – Example: For the input [1, 2, 3, 4, 5], the output should be [2, 3, 4].



The image shows a Python IDE interface. At the top, there is a toolbar with icons for undo, redo, search, run, save, and other standard IDE functions. Below the toolbar, the code editor contains the following Python code:

```
def remove_first_last(lst):  
    return lst[1:-1]  
  
input_list = [1, 2, 3, 4, 5]  
result = remove_first_last(input_list)  
print(result) # Output: [2, 3, 4]
```

Below the code editor, the output console shows the result of the execution:

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



### 5. Get the First n Elements:

Write a Python function that extracts the first n elements from a list.

- Requirements:

- Define a function `get_first_n(lst, n)` that takes a list `lst` and an integer `n` as input and returns the first n elements of the list using slicing.
- Example: For the input `[1, 2, 3, 4, 5]` with `n=3`, the output should be `[1, 2, 3]`.

✓  
0s

```
def get_first_n(lst, n):  
    return lst[:n]  
  
input_list = [1, 2, 3, 4, 5]  
n = 3  
result = get_first_n(input_list, n)  
print(result) # Output: [1, 2, 3]
```

[1, 2, 3]

6. Extract Elements from the End: Write a Python function that extracts the last n elements of a list using slicing.
- Requirements: – Define a function `get_last_n(lst, n)` that takes a list `lst` and an integer `n` as input and returns the last n elements of the list.
  - Example: For the input `[1, 2, 3, 4, 5]` with `n=2`, the output should be `[4, 5]`.




```
def get_last_n(lst, n):  
    return lst[-n:]  
  
input_list = [1, 2, 3, 4, 5]  
n = 2  
result = get_last_n(input_list, n)  
print(result) # Output: [4, 5]
```



[4, 5]

7. Extract Elements in Reverse Order: Write a Python function that extracts a list of elements in reverse order starting from the second-to-last element and skipping one element in between. • Requirements: – Define a function `reverse_skip(lst)` that takes a list `lst` and returns a new list containing every second element starting from the second-to-last, moving backward. – Example: For the input `[1, 2, 3, 4, 5, 6]`, the output should be `[5, 3, 1]`.


```
[36] def reverse_skip(lst):  
    | | return lst[-2::-2]  
input_list = [1, 2, 3, 4, 5, 6]  
result = reverse_skip(input_list)  
print(result) # Output: [5, 3, 1]
```

 [5, 3, 1]


#### 4.3 Exercise on Nested List:

1. Flatten a Nested List: Write a Python function that takes a nested list and flattens it into a single list, where all the elements are in a single dimension. • Requirements: – Define a function `flatten(lst)` that takes a nested list `lst` and returns a flattened version of the list. – Example: For the input `[[1, 2], [3, 4], [5]]`, the output should be `[1, 2, 3, 4, 5]`.

```
def flatten(lst):  
    flat_list = []  
    for sublist in lst:  
        | | for item in sublist:  
        | | | flat_list.append(item)  
    return flat_list  
  
input_list = [[1, 2], [3, 4], [5]]  
result = flatten(input_list)  
print(result) # Output: [1, 2, 3, 4, 5]
```

 [1, 2, 3, 4, 5]

2. Accessing Nested List Elements: Write a Python function that extracts a specific element from a nested list given its indices. • Requirements: – Define a function `access_nested_element(lst, indices)` that takes a nested list `lst` and a list of indices `indices`, and returns the element at that position. – Example: For the input `lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]` with `indices = [1, 2]`, the output should be 6.



```
def access_nested_element(lst, indices):
    element = lst
    for index in indices:
        element = element[index]
    return element

input_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
indices = [1, 2]
result = access_nested_element(input_list, indices)
print(result) # Output: 6
```

 6

3. Sum of All Elements in a Nested List: Write a Python function that calculates the sum of all the numbers in a nested list (regardless of depth). • Requirements: – Define a function `sum_nested(lst)` that takes a nested list `lst` and returns the sum of all the elements. – Example: For the input `[[1, 2], [3, [4, 5]], 6]`, the output should be 21.

```
def sum_nested(lst):
    total = 0
    for item in lst:
        if isinstance(item, list):
            total += sum_nested(item)
        else:
            total += item
    return total
input_list = [[1, 2], [3, [4, 5]], 6]
result = sum_nested(input_list)
print(result) # Output: 21
```

21

4. Remove Specific Element from a Nested List: Write a Python function that removes all occurrences of a specific element from a nested list. • Requirements: – Define a function `remove_element(lst, elem)` that removes `elem` from `lst` and returns the modified list. – Example: For the input `lst = [[1, 2], [3, 2], [4, 5]]` and `elem = 2`, the output should be `[[1], [3], [4, 5]]`.

```
def remove_element(lst, elem):
    if isinstance(lst, list):
        return [remove_element(item, elem) for item in lst if item != elem]
    return lst
input_list = [[1, 2], [3, 2], [4, 5]]
element_to_remove = 2
result = remove_element(input_list, element_to_remove)
print(result) # Output: [[1], [3], [4, 5]]
```

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

5. Find the Maximum Element in a Nested List: Write a Python function that finds the maximum element in a nested list (regardless of depth). • Requirements: – Define a function `find_max(lst)` that takes a nested list `lst` and returns the maximum element. – Example: For the input `[[1, 2], [3, [4, 5]], 6]`, the output should be 6.

0s

+ Code

+ Text

↓

✦

🔗

💬

⚙️

📄

🗑️

⋮

▶

```
def find_max(lst):
    max_val = float('-inf')
    for item in lst:
        if isinstance(item, list):
            max_val = max(max_val, find_max(item))
        else:
            max_val = max(max_val, item)
    return max_val

input_list = [[1, 2], [3, [4, 5]], 6]
result = find_max(input_list)
print(result) # Output: 6
```

↔

6

6. Count Occurrences of an Element in a Nested List: Write a Python function that counts how many times a specific element appears in a nested list. • Requirements: – Define a function `count_occurrences(lst, elem)` that counts the occurrences of `elem` in the nested list `lst`. – Example: For the input `lst = [[1, 2], [2, 3], [2, 4]]` and `elem = 2`, the output should be 3.

```
def count_occurrences(lst, elem):
    count = 0
    for item in lst:
        if isinstance(item, list):
            count += count_occurrences(item, elem)
        elif item == elem:
            count += 1
    return count

input_list = [[1, 2], [2, 3], [2, 4]]
element_to_count = 2
result = count_occurrences(input_list, element_to_count)
print(result) # Output: 3
```

3

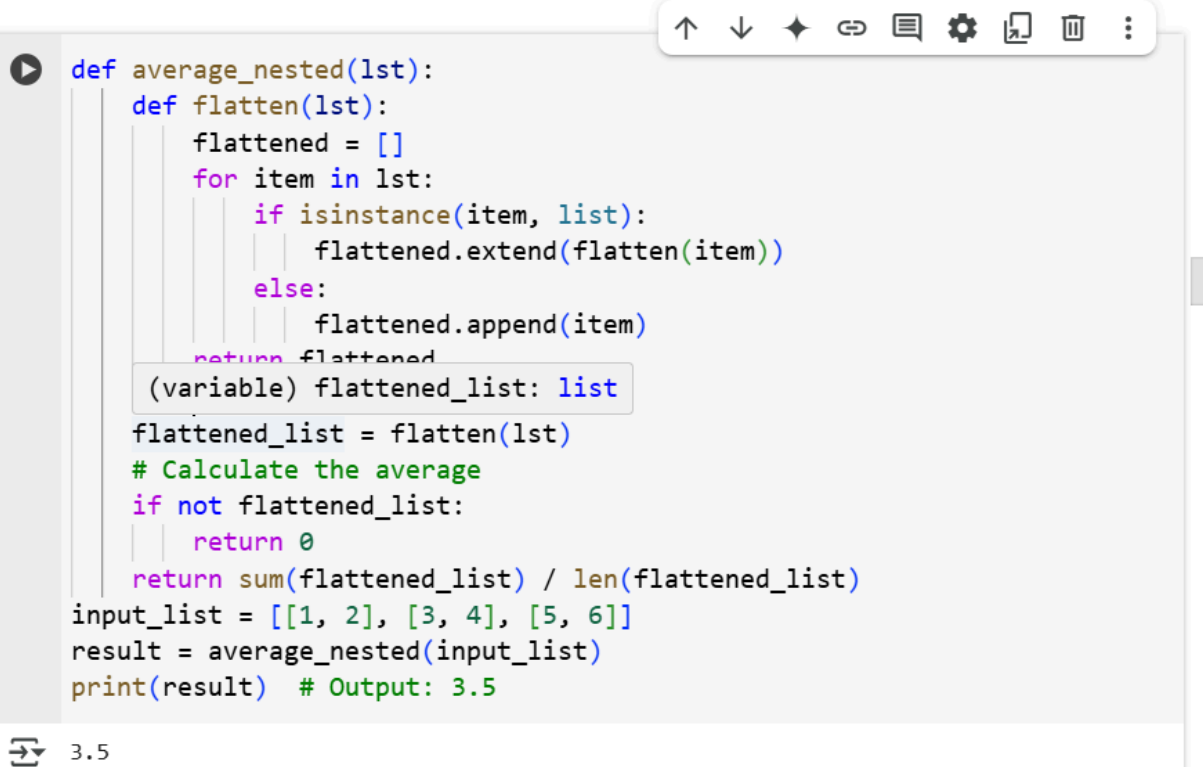
7. Flatten a List of Lists of Lists: Write a Python function that flattens a list of lists of lists into a single list, regardless of the depth. • Requirements: – Define a function `deep_flatten(lst)` that takes a deeply nested list `lst` and returns a single flattened list. – Example: For the input `[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]`, the output should be `[1, 2, 3, 4, 5, 6, 7, 8]`.

```
def deep_flatten(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):
            flat_list.extend(deep_flatten(item))
        else:
            flat_list.append(item)
    return flat_list

input_list = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
result = deep_flatten(input_list)
print(result) # Output: [1, 2, 3, 4, 5, 6, 7, 8]
```

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

8. Nested List Average: Write a Python function that calculates the average of all elements in a nested list. • Requirements: – Define a function `average_nested(lst)` that takes a nested list `lst` and returns the average of all the elements. – Example: For the input `[[1, 2], [3, 4], [5, 6]]`, the output should be 3.5.



```
def average_nested(lst):
    def flatten(lst):
        flattened = []
        for item in lst:
            if isinstance(item, list):
                flattened.extend(flatten(item))
            else:
                flattened.append(item)
        return flattened
    (variable) flattened_list: list
    flattened_list = flatten(lst)
    # Calculate the average
    if not flattened_list:
        return 0
    return sum(flattened_list) / len(flattened_list)
input_list = [[1, 2], [3, 4], [5, 6]]
result = average_nested(input_list)
print(result) # Output: 3.5
```

3.5

## To - Do - NumPy

Problem - 1: Array Creation: Complete the following Tasks:

1. Initialize an empty array with size 2X2.
2. Initialize an all one array with size 4X2.
3. Return a new array of given shape and type, filled with fill value.{Hint: np.full}
4. Return a new array of zeros with same shape and type as a given array.{Hint: np.zeros like}
5. Return a new array of ones with same shape and type as a given array.{Hint: np.ones like}
6. For an existing list new\_list = [1,2,3,4] convert to an numpy array.{Hint: np.array()}

```
import numpy as np

empty_array = np.empty((2, 2))
print("Empty Array:\n", empty_array)

ones_array = np.ones((4, 2))
print("\nAll Ones Array:\n", ones_array)

fill_value = 7
shape = (3, 3)
filled_array = np.full(shape, fill_value)
print("\nFilled Array:\n", filled_array)
```

Zeros Like Array:

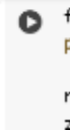
```
[[0 0 0]
 [0 0 0]]
```

Ones Like Array:

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

Converted NumPy Array:

```
[1 2 3 4]
```





Problem - 2: Array Manipulation: Numerical Ranges and Array indexing: Complete the following tasks:

1. Create an array with values ranging from 10 to 49. {Hint:np.arange()}.
2. Create a 3X3 matrix with values ranging from 0 to 8. {Hint:look for np.reshape()}
3. Create a 3X3 identity matrix.{Hint:np.eye()}
4. Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function}
5. Create a 10X10 array with random values and find the minimum and maximum values.
6. Create a zero array of size 10 and replace 5th element with 1.
7. Reverse an array arr = [1,2,0,0,4,0].
8. Create a 2d array with 1 on border and 0 inside.
9. Create a 8X8 matrix and fill it with a checkerboard pattern

```
import numpy as np

array_range = np.arange(10, 50)
print("Array with values from 10 to 49:\n", array_range)

matrix_3x3 = np.arange(9).reshape(3, 3)
print("\n3x3 Matrix with values from 0 to 8:\n", matrix_3x3)

identity_matrix = np.eye(3)
print("\n3x3 Identity Matrix:\n", identity_matrix)

random_array = np.random.random(30)
mean_value = random_array.mean()
print("\nRandom Array of size 30:\n", random_array)
print("Mean of the array:", mean_value)

random_matrix = np.random.random((10, 10))
min_value = random_matrix.min()
max_value = random_matrix.max()
print("\n10x10 Random Matrix:\n", random_matrix)
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)

zero_array = np.zeros(10)
print("\nZero Array with 5th element as 1:\n", zero_array)

arr = np.array([1, 2, 0, 0, 4, 0])
```



```
arr = np.array([1, 2, 0, 0, 4, 0])
reversed_arr = arr[::-1]
print("\nReversed Array:\n", reversed_arr)

border_array = np.ones((5, 5))
border_array[1:-1, 1:-1] = 0
print("\n2D Array with 1 on Border and 0 Inside:\n", border_array)

checkerboard = np.zeros((8, 8), dtype=int)
checkerboard[1::2, ::2] = 1
checkerboard[:, 1::2] = 1
print("\n8x8 Checkerboard Matrix:\n", checkerboard)
```



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

Random Array of size 30:

```
[0.7599497 0.15632468 0.73258158 0.68721366 0.80308236 0.47238684
 0.4140339 0.57311596 0.30828139 0.77748805 0.32789685 0.00705486
 0.96595712 0.79077171 0.02206173 0.68308956 0.15451287 0.29123192
 0.22051171 0.09728489 0.43778672 0.78469416 0.64467593 0.64105995
 0.68680453 0.69781338 0.1210464 0.3988858 0.73063256 0.20809542]
```

Mean of the array: 0.48654420676679105

10x10 Random Matrix:

```
[[0.7393262 0.91584374 0.17712406 0.32068918 0.90978544 0.51796143
 0.25368594 0.25266292 0.54405673 0.37223854]
```

10x10 Random Matrix:



```
[[0.7393262 0.91584374 0.17712406 0.32068918 0.90978544 0.51796143
 0.25368594 0.25266292 0.54405673 0.37223854]
[0.18017113 0.98947281 0.55707686 0.99680133 0.2406144 0.92367676
 0.17363287 0.05737252 0.74125466 0.04769491]
[0.86187838 0.03603877 0.08783723 0.38743338 0.01521772 0.32922128
 0.43940327 0.31626265 0.19214592 0.37088373]
[0.39032226 0.00893025 0.01917244 0.92155363 0.32362492 0.16559162
 0.42298107 0.29460964 0.45004551 0.64959337]
[0.26877815 0.31405203 0.70545265 0.54678219 0.1062361 0.12603789
 0.78080781 0.48272278 0.06294205 0.22072171]
[0.41515306 0.70125928 0.81120581 0.68412981 0.43381671 0.90978527
 0.03496972 0.24665494 0.96094868 0.96810278]
[0.72505286 0.50262639 0.03186497 0.33686824 0.23893155 0.59771455
 0.06500682 0.39245935 0.7811115 0.74332819]
[0.44644531 0.76589398 0.71003768 0.02058996 0.44774211 0.7709023
 0.60915088 0.80201038 0.21867887 0.99911424]
[0.10862187 0.41058663 0.99886578 0.22250682 0.69677089 0.98639412
 0.45908171 0.93989734 0.88644612 0.43656124]
[0.77596267 0.44230798 0.46303621 0.66311104 0.91419702 0.96252728
 0.08284722 0.3129072 0.69756625 0.75010864]]
```

Minimum Value: 0.008930254647709357

Maximum Value: 0.999114239635704

Zero Array with 5th element as 1:

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

Reversed Array:

```
[0 4 0 0 2 1]
```

2D Array with 1 on Border and 0 Inside:

```
[[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 Matrix:

```
[[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: For the following arrays:  $x = \text{np.array}([1,2],[3,5])$  and  $y = \text{np.array}([5,6],[7,8])$ ;  $v = \text{np.array}(9,10)$  and  $w = \text{np.array}(11,12)$ ; Complete all the task using numpy:

1. Add the two array.
2. Subtract the two array.
3. Multiply the array with any integers of your choice.
4. Find the square of each element of the array.
5. Find the dot product between:  $v$  and  $w$  ;  $x$  and  $v$  ;  $x$  and  $y$ .
6. Concatenate  $x$  and  $y$  along row and Concatenate  $v$  and  $w$  along column. {Hint:try `np.concatenate()` or `np.vstack()` functions.
7. Concatenate  $x$  and  $v$ ; if you get an error, observe and explain why did you get the error?

```
[21] import numpy as np

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

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

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

scalar = 3
mul_result = x * scalar
print("\nMultiplication of x with", scalar, ":\n", mul_result)

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

dot_vw = np.dot(v, w)
dot_xv = np.dot(x, v)
dot_xy = np.dot(x, y)
```

[2]

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

concat_xy_row = np.concatenate((x, y), axis=0)
print("\nConcatenation of x and y along rows:\n", concat_xy_row)

concat_vw_col = np.column_stack((v, w))
print("\nConcatenation of v and w along columns:\n", concat_vw_col)

try:
    concat_xv = np.concatenate((x, v), axis=0)
    print("\nConcatenation of x and v:\n", concat_xv)
except ValueError as error:
    print("\nError while concatenating x and v:", error)

# Answer=> "Explanation: x is a 2x2 matrix, while v is a 1D array of s:
# "To concatenate properly, v needs to be reshaped to match x's dimensions"
```



Addition of x and y:

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



Subtraction of x and y:



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

Multiplication of x with 3 :

```
[[ 3  6]
 [ 9 15]]
```

Square of each element in x:

```
[[ 1  4]
 [ 9 25]]
```

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 11]
 [10 12]]
```

Error while concatenating x and v: all the input arrays must have same number of dimensions

Problem - 4: Matrix Operations: • For the following arrays:  $A = \text{np.array}([[3,4],[7,8]])$  and  $B = \text{np.array}([[5,3],[2,1]])$ ; Prove following with Numpy:

1. Prove  $A \cdot A^{-1} = I$ .
2. Prove  $AB \neq BA$ .
3. Prove  $(AB)^T = B^T A^T$ . • Solve the following system of Linear equation using Inverse Methods.  $2x - 3y + z = -1$   $x - y + 2z = -3$   $3x + y - z = 9$  {Hint: First use Numpy array to represent the equation in Matrix form. Then Solve for:  $AX = B$ } • Now: solve the above equation using `np.linalg.inv` function. {Explore more about "linalg" function of Numpy}

```
import numpy as np

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

A_inv = np.linalg.inv(A)
identity_matrix = np.dot(A, A_inv)

print("A * A-1:\n", identity_matrix)
print("Expected Identity Matrix:\n", np.eye(2))

AB = np.dot(A, B)
BA = np.dot(B, A)

print("\nAB:\n", AB)
print("\nBA:\n", BA)
print("\nAre AB and BA equal?", np.array_equal(AB, BA))

AB_T = np.transpose(AB)
BT_AT = np.dot(B.T, A.T)

print("\n(AB)T:\n", AB_T)
print("\nBT * AT:\n", BT_AT)
print("\nIs (AB)T = BT * AT?", np.array_equal(AB_T, BT_AT))
```



```

▶ A_matrix = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])
  B_matrix = np.array([-1, -3, 9])

A_inv_matrix = np.linalg.inv(A_matrix)
X_solution = np.dot(A_inv_matrix, B_matrix)

print("\nSolution for x, y, z using inverse method:\n", X_solution)
X_solution_np = np.linalg.solve(A_matrix, B_matrix)
print("\nSolution for x, y, z using np.linalg.solve:\n", X_solution_np)

```

```

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

```

Expected Identity Matrix:

```

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

```

AB:

```

[[23 13]
 [51 29]]

```

BA:

```

[[36 44]
 [13 16]]

```

Are AB and BA equal? False



```
(AB)T:  
[[23 51]  
 [13 29]]
```

```
BT * AT:  
[[23 51]  
 [13 29]]
```

```
Is (AB)T = BT * AT? True
```

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

```
Solution for x, y, z using np.linalg.solve:  
[ 2.  1. -2.]
```

10.2 Experiment: How Fast is Numpy? In this exercise, you will compare the performance and implementation of operations using plain Python lists (arrays) and NumPy arrays. Follow the instructions:

1. Element-wise Addition: • Using Python Lists, perform element-wise addition of two lists of size 1,000,000. Measure and Print the time taken for this operation. • Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.
2. Element-wise Multiplication • Using Python Lists, perform element-wise multiplication of two lists of size 1,000,000. Measure and Print the time taken for this operation. • Using

2. Element-wise Multiplication • Using Python Lists, perform element-wise multiplication of two lists of size 1,000,000. Measure and Print the time taken for this operation. • Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.
3. Dot Product • Using Python Lists, compute the dot product of two lists of size 1,000,000. Measure and Print the time taken for this operation. • Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.
4. Matrix Multiplication • Using Python lists, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation. • Using NumPy arrays, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation

```
import numpy as np
import time

size = 1_000_000

list_a = list(range(size))
list_b = list(range(size))

start = time.time()
result_add_list = [list_a[i] + list_b[i] for i in range(size)]
end = time.time()
print(f"Python List Addition Time: {end - start:.5f} seconds")
```

```
print(f"Python List Addition Time: {end - start:.5f} seconds")

array_a = np.arange(size)
array_b = np.arange(size)

start = time.time()
result_add_array = array_a + array_b
end = time.time()
print(f"NumPy Array Addition Time: {end - start:.5f} seconds")

start = time.time()
result_mul_list = [list_a[i] * list_b[i] for i in range(size)]
end = time.time()
print(f"Python List Multiplication Time: {end - start:.5f} seconds")

start = time.time()
result_mul_array = array_a * array_b
end = time.time()
print(f"NumPy Array Multiplication Time: {end - start:.5f} seconds")

start = time.time()
dot_product_list = sum(list_a[i] * list_b[i] for i in range(size))
end = time.time()
print(f"Python List Dot Product Time: {end - start:.5f} seconds")

start = time.time()
dot product array = np.dot(array a, array b)
```

```

start = time.time()
dot_product_array = np.dot(array_a, array_b)
end = time.time()
print(f"NumPy Array Dot Product Time: {end - start:.5f} seconds")

matrix_size = 1000

matrix_a = [[i + j for j in range(matrix_size)] for i in range(matrix_size)]
matrix_b = [[i - j for j in range(matrix_size)] for i in range(matrix_size)]
result_matrix = [[0] * matrix_size for _ in range(matrix_size)]

start = time.time()
for i in range(matrix_size):
    for j in range(matrix_size):
        for k in range(matrix_size):
            result_matrix[i][j] += matrix_a[i][k] * matrix_b[k][j]
end = time.time()
print(f"Python List Matrix Multiplication Time: {end - start:.5f} seconds")

np_matrix_a = np.random.rand(matrix_size, matrix_size)
np_matrix_b = np.random.rand(matrix_size, matrix_size)

start = time.time()
np_result_matrix = np.dot(np_matrix_a, np_matrix_b)
end = time.time()
print(f"NumPy Array Matrix Multiplication Time: {end - start:.5f} seconds")

```

```

end = time.time()
print(f"NumPy Array Matrix Multiplication Time: {end - start:.5f} seconds")

```

```

Python List Addition Time: 0.09680 seconds
NumPy Array Addition Time: 0.01529 seconds
Python List Multiplication Time: 0.16004 seconds
NumPy Array Multiplication Time: 0.01769 seconds
Python List Dot Product Time: 0.09503 seconds
NumPy Array Dot Product Time: 0.00188 seconds
Python List Matrix Multiplication Time: 425.32622 seconds
NumPy Array Matrix Multiplication Time: 0.06172 seconds

```