Section I- Python

Name: Renuka Dhakal

Student ID: 2330485

Group: L6CG20

Task-1

Exercise on Functions:

Task - 1: 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:
5. Functions: Define at least one function to perform the conversion.
6. Error Handling: Use try-except blocks to handle invalid input (e.g., non-numeric values).
7. User Input: Prompt the user to select the conversion type and input the value.
8. Docstrings: Include a docstring in your function to describe its purpose, parameters, and return value. • Conversion Options:
9. Length: – Convert meters (m) to feet (ft). – Convert feet (ft) to meters (m).
10. Weight: – Convert kilograms (kg) to pounds (lbs). – Convert pounds (lbs) to kilograms (kg).
11. Volume: – Convert liters (L) to gallons (gal). – Convert gallons (gal) to liters (L).

```
# Create a Python program that converts between different units of measurement.


# Functions to convert units
def length_converter(val, unit):
  """
  Convert length into feet or meter

  Parameters: val is any numeric value to be converted. (float) and unit is the string denoting unit type (string).
  Returns: Result value(float)

  """

  if unit == "meter":
    return val * 3.280
  elif unit == "feet":
    return val / 3.280
  else:
    return ValueError("Invalid Unit entered!")

def weight_converter(val, unit):
  """
  Convert weight into pounds or kg

  Parameters: val is any numeric value to be converted. (float) and unit is the string denoting unit type (string).
  Returns: Result value(float)

  """

  if unit == "kg":
    return val / 2.205
  elif unit == "pounds":
    return val * 2.205
  else:
    return ValueError("Invalid Unit entered!")


def volume_converter(val, unit):
  """
  Convert volume into gallons or liters
  Parameters: val is any numeric value to be converted. (float) and unit is the string denoting unit type (string).
  Returns: Result value(float)

  """

  if unit == "gallons":
    return val * 0.265
  elif unit == "liters":
    return val / 0.265
  else:
    return ValueError("Invalid Unit entered!")




# Main function for the task

def main():
  print("""
  Unit Converter
  Choose a conversion type:
  1. Length (meters <==> feet)
  2. Weight (kilograms <==> pounds)
  3. Volume (liters <==> gallons)
  """)

  try:
    choice = int(input("Enter your choice for conversion: "))

    if choice == 1:
      unit = input("Enter the unit of length you want to convert to (meters/feet): ")
      val = float(input(f"Enter the value to be converted to {unit}: "))
      result = length_converter(val, unit)
      print(f"The converted value of length is {result} {unit}.")

    elif choice == 2:
      unit = input("Enter the unit of weight you want to convert to (kg/pounds): ")
      val = float(input(f"Enter the value to be converted to {unit}: "))
      result = weight_converter(val, unit)
      print(f"The converted value is {result} {unit}.")

    elif choice == 3:
      unit = input("Enter the unit of volume you want to convert to (liters/gallons): ")
      val = float(input(f"Enter the value to be converted to {unit}: "))
      result = volume_converter(val, unit)
      print(f"The converted value is {result} {unit}.")

    else:
      print("Invalid choice entered!")
      return

  except ValueError as e:
    print(f"Error {e}")

  except Exception as e:
    print(f"Error {e}")

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

```

  Unit Converter
  Choose a conversion type:
  1. Length (meters <==> feet)
  2. Weight (kilograms <==> pounds)
  3. Volume (liters <==> gallons)

Enter your choice for conversion: 1
Enter the unit of length you want to convert to (meters/feet): feet
Enter the value to be converted to feet: 50
The converted value of length is 15.24390243902439 feet.
```

Task 2

Create a Python program that performs various mathematical operations on a list of numbers. • The Program should:

1. Prompt the user to choose an operation (e.g., find the sum, average, maximum, or minimum of the numbers).
2. Ask the user to input a list of numbers (separated by spaces).
3. Perform the selected operation and display the result.
4. Handle potential errors, such as invalid input or empty lists. • Requirements:
5. Functions: Define at least one function for each operation (sum, average, maximum, minimum).
6. Error Handling: Use try-except blocks to handle invalid input (e.g., non-numeric values or empty lists).
7. User Input: Prompt the user to select the operation and input the list of numbers.
8. Docstrings: Include a docstring in each function to describe its purpose, parameters, and return value.

```
# Create a Python program that performs various mathematical operations on a list of numbers.

# Functions to perform operations of number list
def calculate_sum(numbers):
  """
  Calculate the suom of the numbers in list

  PArameter is list of numbers(float or int)

  Returns sum of numbers(float)
  """

  return sum(numbers)


def calculate_average(numbers):
  """
  Calculate the average of the numbers in list

  Parameter is list of numbers(float or int)

  Returns average of numbers(float)
  """

  return sum(numbers) / len(numbers)

def get_maximum(numbers):
  """
  Get the maximum number in the list

  Parameter is list of numbers(float or int)

  Returns maximum number(float)
  """

  return max(numbers)


def get_minimum(numbers):
  """
  Get the minimum number in the list

  Parameter is list of numbers(float o r int)

  Returns minimum number(float)
  """

  return min(numbers)


# Main function for task 2
def main():
  print("""
  MAthematical Operations in list
  Choose an operation:
  1. Sum
  2. Average
  3. Maximum
  4. Minimum
  """)

  try:
    choice = int(input("Enter your choice: "))
    numbers_list = input("Enter the list of numbers separated by spaces: ").split()
    numbers_list = [float(num) for num in numbers_list]

    if choice == 1:
      result = calculate_sum(numbers_list)
      operation = "Sum"

    elif choice == 2:
      result = calculate_average(numbers_list)
      operation = "Average"

    elif choice == 3:
      result = get_maximum(numbers_list)
      operation = "Maximum"

    elif choice == 4:
      result = get_minimum(numbers_list)
      operation = "Minimum"

    else:
      print("Invalid choice entered!")
      return

    print(f"The {operation} of the numbers in provided list is {result}.")

  except ValueError as e:
    print(f"Error {e}")

  except Exception as e:
    print(f"Error {e}")

  finally:
    print("Task Ended for Mathematical operations in list!")

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

```
  MAthematical Operations in list
  Choose an operation:
  1. Sum
  2. Average
  3. Maximum
  4. Minimum

Enter your choice: 3
Enter the list of numbers separated by spaces: 7 9 6 9 56
The Maximum of the numbers in provided list is 56.0.
Task Ended for Mathematical operations in list!
```

Task 3 - Exercises on list manipulation

a. 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]

lst = [1, 2, 3, 4, 5, 6]
result = extract_every_other(lst)
print(f"{lst} List after extracting every other element is {result}.")
```

```
[1, 2, 3, 4, 5, 6] List after extracting every other element is [1, 3, 5].
```

b. 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]

lst = [1, 2, 3, 4, 5, 6]
start = 2
end = 4
result = get_sublist(lst, start, end)
print(f"{lst} after getting its sublist from index {start} to {end} is {result}.")
```

```
[1, 2, 3, 4, 5, 6] after getting its sublist from index 2 to 4 is [3, 4, 5].
```

c. 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].

```
def reverse_list(lst):
  return lst[::-1]

lst = [1, 2, 3, 4, 5]
result = reverse_list(lst)
print(f"{lst} after reversing is {result}.")
```

```
[1, 2, 3, 4, 5] after reversing is [5, 4, 3, 2, 1].
```

d. 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].

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

lst = [1, 2, 3, 4, 5]
result = remove_first_last(lst)
print(f"{lst} after removing first and last element is {result}.")
```

```
[1, 2, 3, 4, 5] after removing first and last element is [2, 3, 4].
```

e. 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].

```
def get_first_n(lst, n):
  return lst[:n]

lst = [1, 2, 3, 4, 5]
n = 3
```

```
result = get_first_n(lst, n)
print(f"{lst} after getting first {n} elements is {result}.")
```

```
[1, 2, 3, 4, 5] after getting first 3 elements is [1, 2, 3].
```

f. 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:]

lst = [1, 2, 3, 4, 5]
n = 2
result = get_last_n(lst, n)
print(f"{lst} after getting last {n} elements is {result}.")
```

```
[1, 2, 3, 4, 5] after getting last 2 elements is [4, 5].
```

g. 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].

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

lst = [1, 2, 3, 4, 5, 6]
result = reverse_skip(lst)
print(f"{lst} after reversing and skipping is {result}.")
```

```
[1, 2, 3, 4, 5, 6] after reversing and skipping is [5, 3, 1].
```

Task 4 - Exercises on nested list

a. 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 item in lst:
    if isinstance(item, list):
      flat_list.extend(flatten(item))
    else:
      flat_list.append(item)
  return flat_list


lst = [[1, 2], [3, 4], [5]]
result = flatten(lst)
print(f"{lst} after flattening is {result}.")
```

```
[[1, 2], [3, 4], [5]] after flattening is [1, 2, 3, 4, 5].
```

b. 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

  try:
    for index in indices:
      element = element[index]

    return element

  except(IndexError, TypeError):
    return "Invalid Indices"

lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
indices = [1, 2]
result = access_nested_element(lst, indices)
print(f"{lst} accessing element at indices {indices} is {result}.")
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]] accessing element at indices [1, 2] is 6.
```

c. 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):
  return sum(flatten(lst))

lst = [[1, 2], [3, [4, 5]], 6]
result = sum_nested(lst)
print(f"{lst} sum of all elements is {result}.")
```

```
[[1, 2], [3, [4, 5]], 6] sum of all elements is 21.
```

d. 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):
  result = []

  for item in lst:
    if isinstance(item, list):
      result.append(remove_element(item, elem))
    elif item != elem:
      result.append(item)

  return result

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

```
elem = 2
result = remove_element(lst, elem)
print(f"{lst} after removing element {elem} is {result}.")
```

```
[[1, 2], [3, 2], [4, 5]] after removing element 2 is [[1], [3], [4, 5]].
```

e. 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.

```
def find_max(lst):
  return max(flatten(lst))

lst = [[1, 2], [3, [4, 5]], 6]
result = find_max(lst)
print(f"{lst} maximum element is {result}.")
```

```
[[1, 2], [3, [4, 5]], 6] maximum element is 6.
```

f. 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_occourances(lst, elem):
  return flatten(lst).count(elem)

lst = [[1, 2], [2, 3], [2, 4]]
elem = 2
result = count_occourances(lst, elem)
print(f"{lst} count of element {elem} is {result}.")
```

```
[[1, 2], [2, 3], [2, 4]] count of element 2 is 3.
```

g. 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):
  return flatten(lst)

lst = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
result = deep_flatten(lst)
print(f"{lst} after deep flattening is {result}.")
```

```
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]] after deep flattening is [1, 2, 3, 4, 5, 6, 7, 8].
```

h. 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):
  return sum(flatten(lst)) / len(flatten(lst))


lst = [[1, 2], [3, 4], [5, 6]]
result = average_nested(lst)
print(f"{lst} average of all elements is {result}.")
```

```
[[1, 2], [3, 4], [5, 6]] average of all elements is 3.5.
```

Section II - Numpy

```
import numpy as np
import time
```

Task 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()}

```
# 1. Initialize an empty array with size 2X2.
emt_array = np.empty((2, 2))
print("1. Empty array with size 2X2:")
print(emt_array,"\n")


# 2. Initialize an all one array with size 4X2.
ones_Array = np.ones((4,2))
print("2. All ones array with size 4X2:")
print(ones_Array, "\n")


# 3. Return a new array of given shape and type, filled with fill value.
filled_Array = np.full((4,3), 7)
print("3. Array of given shape and type, filled with fill value:")
print(filled_Array, "\n")


# 4. Return a new array of zeros with same shape and type as a given array.
sample_array = np.array([[1, 5, 8], [3, 9, 2]])
zeros_array = np.zeros_like(sample_array)
print("4. Array of zeros with same shape and type as a given sample array:")
print(zeros_array, "\n")


# 5. Return a new array of ones with same shape and type as a given array.
ones_array = np.ones_like(sample_array)
print("5. Array of ones with same shape and type as a given sample array:")
print(ones_array, "\n")

# 6. For an existing list new_list = [1,2,3,4] convert to an numpy array.
new_list = [1, 2, 3, 4]
numpy_array = np.array(new_list)
print("6. Numpy array of the provided new_list array:")
print(numpy_array, "\n")
```

```
1. Empty array with size 2X2:
[[2.4110926e-316 0.0000000e+000]
 [1.5810101e-322 6.8490785e-310]]
```

```
2. All ones array with size 4X2:
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]

3. Array of given shape and type, filled with fill value:
[[7 7 7]
 [7 7 7]
 [7 7 7]
 [7 7 7]]

4. Array of zeros with same shape and type as a given sample array:
[[0 0 0]
 [0 0 0]]

5. Array of ones with same shape and type as a given sample array:
[[1 1 1]
 [1 1 1]]

6. Numpy array of the provided new_list array:
[1 2 3 4]
```

Task 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.arrange()}.
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.

```
# 1. Create an array with values ranging from 10 to 49.
ranged_array = np.arange(10, 50)
print(f"1. Array from range 10 to 49:\n {ranged_array}.\n")


# 2. Create a 3X3 matrix with values ranging from 0 to 8.
matrix3x3 = np.arange(9).reshape(3, 3)
print(f"2. 3X3 matrix with values ranging from 0 to 8:\n {matrix3x3} \n")


# 3. Create a 3X3 identity matrix.
i_matrix = np.eye(3)
print(f"3. 3X3 identity matrix:\n {i_matrix} \n")


# 4. Create a random array of size 30 and find the mean of the array.
random_array = np.random.random(30)
print(f"4. Random array of size 30:\n {random_array} \n")


# 5. Create a 10X10 array with random values and find the minimum and maximum values.
random_matrix = np.random.random((10, 10))
min_value = np.min(random_matrix)
max_value = np.max(random_matrix)
print(f"5. 10X10 array with random values:\n {random_matrix} \n")
print(f"Minimum value: {min_value} \n")
print(f"Maximum value: {max_value} \n\n")


# 6. Create a zero array of size 10 and replace 5th element with 1.
zero_array = np.zeros(10)
print(f"6. Zero array of size 10:\n {zero_array} \n")
zero_array[4] = 1
print(f"After replacing 5th element with 1:\n {zero_array} \n")
```

```
# 7. Reverse an array arr = [1,2,0,0,4,0].
arr = np.array([1, 2, 0, 0, 4, 0])
reversed_arr = arr[::-1]
print(f"7.{arr} Reversed array:\n {reversed_arr} \n")


# 8. Create a 2d array with 1 on border and 0 inside.
border_1 = np.ones((5, 5))
border_1[1:-1, 1:-1] = 0
print(f"8. 2D array with 1 on border and 0 inside:\n {border_1} \n")


# 9. Create a 8X8 matrix and fill it with a checkerboard pattern.
checkboard = np.zeros((8, 8))
checkboard[1::2, ::2] = 1
checkboard[::2, 1::2] = 1
print(f"9. 8X8 matrix with checkerboard pattern:\n {checkboard} \n")
```

```
1. Array from range 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].

2. 3X3 matrix with values ranging from 0 to 8:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]

3. 3X3 identity matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

4. Random array of size 30:
 [0.59679127 0.76584715 0.82509824 0.43194341 0.64617171 0.81469974
 0.98186956 0.19951201 0.23470524 0.60850366 0.87259932 0.88902566
 0.13839571 0.51683573 0.00649225 0.16924948 0.74985107 0.41200929
 0.92255763 0.33065566 0.8761316  0.92332697 0.24408962 0.1010944
 0.5102976  0.6321507  0.10007299 0.46575547 0.58366064 0.60398386]

5. 10X10 array with random values:
 [[0.07732917 0.98228841 0.65188144 0.39764778 0.70912489 0.41287543
  0.94956422 0.92312466 0.83060689 0.10537432]
 [0.68579344 0.7582697  0.44239119 0.2047676  0.90044885 0.9995739
  0.79399687 0.21918239 0.40110024 0.25946841]
 [0.55424504 0.02989911 0.03850869 0.82251168 0.65275065 0.35655523
  0.37749823 0.17638237 0.8010185  0.28532841]
 [0.23413635 0.34190424 0.29001585 0.67748311 0.48117552 0.38169435
  0.77362618 0.05382408 0.11186118 0.63972554]
 [0.04760952 0.18594519 0.71556366 0.54942785 0.67376998 0.21003877
  0.44754455 0.78021989 0.0714249  0.73084292]
 [0.91035233 0.48298972 0.02724422 0.12042275 0.64037965 0.99791932
  0.41164355 0.72244159 0.42439152 0.88154616]
 [0.48168901 0.42163349 0.33446839 0.58200998 0.67495017 0.51167562
  0.6659346  0.55985972 0.99979461 0.20711459]
 [0.68142044 0.49869458 0.53428077 0.93580181 0.73229047 0.72013084
  0.6267956  0.22169476 0.87126476 0.52532419]
 [0.66165813 0.57440262 0.18930358 0.51689264 0.34012132 0.7503427
  0.82158435 0.77857358 0.86597201 0.99436513]
 [0.19968616 0.37621007 0.55574432 0.50223598 0.26528154 0.61015236
  0.19633946 0.01456496 0.91275389 0.21226876]]

Minimum value: 0.01456496056002854

Maximum value: 0.9997946073165318


6. Zero array of size 10:
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

After replacing 5th element with 1:
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

7.[1 2 0 0 4 0] Reversed array:
 [0 4 0 0 2 1]

8. 2D array with 1 on border and 0 inside:
```

Task 3: Array Operations

For the following arrays: x = np.array([[1,2],[3,5]]) and y = np.array([[5,6],[7,8]]);

v = np.array([9,10]) and w = 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?

```
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 the two array.
add_result = x + y
print(f"1. Addition of x and y:\n {add_result} \n")


# 2. Subtract the two array.
sub_result = x - y
print(f"2. Subtraction of x and y:\n {sub_result} \n")


# 3. Multiply the array with any integers of your choice.
prod_result = 3 * x
print(f"3. Multiplying x by 3:\n {prod_result} \n")


# 4. Find the square of each element of the array.
square_y = np.square(y)
print(f"4. Square of y:\n {square_y} \n")


# 5. Finding the dot products
dot_xy = np.dot(x,y)
dot_xv = np.dot(x, v)
dot_vw = np.dot(v, w)
print(f"5. The dot product between v and w is {dot_vw}; x and v is {dot_xv}; and x & y is{dot_xy}\n")


# 6. Concatenate x(and)y along row and Concatenate v(and)w along column.
conc_xy = np.concatenate((x, y), axis = 0)
conc_vw = np.column_stack((v, w))
print(f"6. Concatenation of x and y along rows is\n\n{conc_xy} \n\n and of v and w along column is {conc_vw}\n")
```

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

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

3. Multiplying x by 3:
 [[ 3  6]
 [ 9 15]]

4. Square of y:
 [[25 36]
 [49 64]]

5. The dot product between v and w is 219; x and v is [29 77]; and x & y is[[19 22]
 [50 58]]

6. Concatenation of x and y along rows is

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

and of v and w along column is [[ 9 11]
 [10 12]]
```

```
# 7. Concatenate x(and)v; if you get an error, observe and explain why did you get the error?
conc_xv = np.concatenate((x, v), axis = 0)
print(f"7. Concatenation of x and v: {conc_xv}")


"""
x has shape (2,2) and v has shape (2,). The dimensions do not match along the chosen axis.
"""
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-ffb58c0ce904> in <cell line: 0>()
      1 # 7. Concatenate x(and)v; if you get an error, observe and explain why did you get the error?
----> 2 conc_xv = np.concatenate((x, v), axis = 0)
      3 print(f"7. Concatenation of x and v: {conc_xv}")
      4
      5

ValueError: 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 dimension(s)
```

Next steps: Explain error

Task 4: Matrix Operations

For the following arrays:

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

Prove following with Numpy:

1. Prove A.A−1 = I.
2. Prove AB ≠ 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}

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


# Proving A.A-1 = I

A_inverse = np.linalg.inv(A)

identity = np.dot(A, A_inverse)

print(f"1. Proving A.A-1 = I for \n{A} is\n{identity}\n\n")



# Proving AB != BA
AB = np.dot(A,B)
BA = np.dot(B,A)

print(f"2. Proved that AB != BA since AB = \n{AB}\n and BA =\n{BA}\n\n")


# Prove (AB)^T= B^T . A^T
```

```
AB_t = np.transpose(AB)
B_t = np.transpose(B)
A_t = np.transpose(A)

Bt_dot_At = np.dot(B_t, A_t)

print(f"3. Proved that (AB)^T = B^T . A^T since (AB)^T = \n{AB_t}\n and B^T . A^T = \n{Bt_dot_At}\n\n")
```

```
1. Proving A.A-1 = I for
[[3 4]
 [7 8]] is
[[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]


2. Proved that AB != BA since AB =
[[23 13]
 [51 29]]
 and BA =
[[36 44]
 [13 16]]


3. Proved that (AB)^T = B^T . A^T since (AB)^T =
[[23 51]
 [13 29]]
 and B^T . A^T =
[[23 51]
 [13 29]]
```

```
# Solving for linear equation using matrix method in numpy for
# 2x − 3y + z = −1

# x − y + 2z = −3

# 3x + y − z = 9

le_A = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])
le_B = np.array([-1, -3, 9])

# If A-1 exists then, X = A-1 . B

le_A_inverse = np.linalg.inv(le_A)
le_X = np.dot(le_A_inverse, le_B)

print(f"The [x, y, z] of the system is\n{le_X}\n")
```

```
The [x, y, z] of the system is
[ 2.  1. -2.]
```

Task 5: Experiment: How Fast is Numpy?

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.

```
size = 1000000
list1 = list(range(size))
list2 = list(range(size))

# Measuring time for element addition
start_time = time.time()
list_result = []
for i in range(size):
  list_result.append(list1[i] + list2[i])
end_time = time.time()


print(f"Time taken for python list addition is {end_time - start_time} seconds.\n")
```

```
# Now measuring time for numpy addition
array1 = np.arange(size)
array2 = np.arange(size)

start_time = time.time()
array_result = array1 + array2
end_time = time.time()

print(f"Time taken for numpy array addition is {end_time - start_time} seconds.\n")
```

```
Time taken for python list addition is 0.1724400520324707 seconds.

Time taken for numpy array addition is 0.004145622253417969 seconds.
```

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

```
# Measuring time for python list multiplication
start_time = time.time()
list_result = []
for i in range(size):
  list_result.append(list1[i] * list2[i])
end_time = time.time()


print(f"Time taken for python list multiplication is {end_time - start_time} seconds.\n")

# Measuring time for numpy array multiplication
start_time = time.time()
array_result = array1 * array2
end_time = time.time()

print(f"Time taken for numpy array multiplication is {end_time - start_time} seconds.\n")
```

```
Time taken for python list multiplication is 0.19631671905517578 seconds.

Time taken for numpy array multiplication is 0.005066394805908203 seconds.
```

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.

```
# Measuring time for dot product in python list
start_time = time.time()
dot_products = []
for i in range(size):
  dot_products.append(list1[i] * list2[i])
dot_product_result = sum(dot_products)
end_time = time.time()

print(f"Time taken for python list dot product is {end_time - start_time} seconds.\n")


# Measuring time for dot products using numpy
start_time = time.time()
dot_product = np.dot(array1, array2)
end_time = time.time()

print(f"Time taken for numpy array dot product is {end_time - start_time} seconds.\n")
```

```
Time taken for python list dot product is 0.21472716331481934 seconds.

Time taken for numpy array dot product is 0.0020782947540283203 seconds.
```

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.

```
size = 1000
# Measuring time taken for matrix multiplication using python list
mat_1 = []
mat_2 = []

for i in range(size):
  row = []
  for j in range(size):
    row.append(i*size +j)
  mat_1.append(row)

for i in range(size):
  row = []
  for j in range(size):
    row.append(i*size+j)
  mat_2.append(row)

start_time = time.time()
result_matrix = []
for i in range(size):
  for j in range(size):
    sum = 0
    for k in range(0, size):
      sum += mat_1[i][k] * mat_2[k][j]
    result_matrix.append(sum)
end_time = time.time()

print(f"Time taken for python list matrix multiplication is {end_time - start_time} seconds.\n")




# Measuring time taken for matrix multiplication using numpy
mat_1 = np.arange(size * size).reshape(size, size)
mat_2 = np.arange(size * size).reshape(size, size)

start_time = time.time()
result_matrix = np.dot(mat_1, mat_2)
end_time = time.time()

print(f"Time taken for numpy matrix multiplication is {end_time - start_time} seconds.\n")
```

```
Time taken for python list matrix multiplication is 359.1364390850067 seconds.

Time taken for numpy matrix multiplication is 1.2152631282806396 seconds.
```