Exercise on Functions

Task 1: Unit Conversion Program

Create a Python program that converts between different units of measurement (length, weight, volume). The program should:

Prompt the user to choose the type of conversion.

Ask the user to input the value to be converted.

Perform the conversion and display the result.

Handle potential errors (e.g., invalid input).

```
def unit_converter():
    """
    Converts between different units of measurement.
    Options: length (meters to feet, feet to meters),
             weight (kilograms to pounds, pounds to kilograms),
             volume (liters to gallons, gallons to liters).
    """
    try:
        # Prompt user for conversion type
        print("Choose conversion type:")
        print("1. Length")
        print("2. Weight")
        print("3. Volume")
        choice = int(input("Enter 1, 2, or 3: "))

        # Prompt user for value to convert
        value = float(input("Enter the value to convert: "))

        # Perform conversion based on user choice
        if choice == 1:
            print("1. Meters to Feet")
            print("2. Feet to Meters")
            sub_choice = int(input("Enter 1 or 2: "))
            if sub_choice == 1:
                result = value * 3.28084
                print(f"{value} meters = {result} feet")
            elif sub_choice == 2:
                result = value / 3.28084
                print(f"{value} feet = {result} meters")
            else:
                print("Invalid choice.")
        elif choice == 2:
            print("1. Kilograms to Pounds")
            print("2. Pounds to Kilograms")
            sub_choice = int(input("Enter 1 or 2: "))
            if sub_choice == 1:
                result = value * 2.20462
                print(f"{value} kg = {result} lbs")
            elif sub_choice == 2:
                result = value / 2.20462
                print(f"{value} lbs = {result} kg")
            else:
                print("Invalid choice.")
        elif choice == 3:
            print("1. Liters to Gallons")
            print("2. Gallons to Liters")
            sub_choice = int(input("Enter 1 or 2: "))
            if sub_choice == 1:
                result = value * 0.264172
                print(f"{value} liters = {result} gallons")
            elif sub_choice == 2:
                result = value / 0.264172
                print(f"{value} gallons = {result} liters")
            else:
                print("Invalid choice.")
        else:
            print("Invalid choice.")
    except ValueError:
        print("Invalid input. Please enter numeric values.")
```

```
# Call the function
unit_converter()
```

```
Choose conversion type:
1. Length
2. Weight
3. Volume
Enter 1, 2, or 3: 3
Enter the value to convert: 7
1. Liters to Gallons
2. Gallons to Liters
Enter 1 or 2: 2
7.0 gallons = 26.497887739805883 liters
```

Task 2: Mathematical Operations on a List

Create a Python program that performs various mathematical operations on a list of numbers (sum, average, maximum, minimum).

```
def sum_list(numbers):
    """Returns the sum of a list of numbers."""
    return sum(numbers)

def average_list(numbers):
    """Returns the average of a list of numbers."""
    return sum(numbers) / len(numbers)

def max_list(numbers):
    """Returns the maximum value in a list of numbers."""
    return max(numbers)

def min_list(numbers):
    """Returns the minimum value in a list of numbers."""
    return min(numbers)

def math_operations():
    """
    Performs mathematical operations on a list of numbers.
    Options: sum, average, maximum, minimum.
    """
    try:
        # Prompt user for list of numbers
        numbers = list(map(float, input("Enter a list of numbers separated by spaces: ").split()))

        # Prompt user for operation
        print("Choose operation:")
        print("1. Sum")
        print("2. Average")
        print("3. Maximum")
        print("4. Minimum")
        choice = int(input("Enter 1, 2, 3, or 4: "))

        # Perform operation based on user choice
        if choice == 1:
            result = sum_list(numbers)
            print(f"Sum: {result}")
        elif choice == 2:
            result = average_list(numbers)
            print(f"Average: {result}")
        elif choice == 3:
            result = max_list(numbers)
            print(f"Maximum: {result}")
        elif choice == 4:
            result = min_list(numbers)
            print(f"Minimum: {result}")
        else:
            print("Invalid choice.")
    except ValueError:
        print("Invalid input. Please enter numeric values.")

# Call the function
math_operations()
```

```
Enter a list of numbers separated by spaces: 5
Choose operation:
1. Sum
2. Average
3. Maximum
```

```
4. Minimum
Enter 1, 2, 3, or 4: 1
Sum: 5.0
```

Exercise on List Manipulation

Task 1: Extract Every Other Element

Write a Python function that extracts every other element from a list, starting from the first element.

```
def extract_every_other(lst):
    """Extracts every other element from a list, starting from the first element."""
    return lst[::2]

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

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

Task 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.

```
def get_sublist(lst, start, end):
    """Returns a sublist from a given list, starting from start index and ending at end index."""
    return lst[start:end+1]
print(get_sublist([1, 2, 3, 4, 5, 6], 2, 4))
```

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

Task 3: Reverse a List Using Slicing

Write a Python function that reverses a list using slicing.

```
def reverse_list(lst):
    """Reverses a list using slicing."""
    return lst[::-1]
print(reverse_list([1, 2, 3, 4, 5]))
```

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

Task 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.

```
def remove_first_last(lst):
    """Removes the first and last elements of a list and returns the resulting sublist."""
    return lst[1:-1]
print(remove_first_last([1, 2, 3, 4, 5]))
```

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

Task 5: Get the First n Elements

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

```
def get_first_n(lst, n):
    """Extracts the first n elements from a list."""
    return lst[:n]
print(get_first_n([1, 2, 3, 4, 5], 3))
```

```
[1, 2, 3]
```

Task 6: Extract Elements from the End

Write a Python function that extracts the last n elements of a list using slicing.

```
def get_last_n(lst, n):
    """Extracts the last n elements of a list using slicing."""
    return lst[-n:]
print(get_last_n([1, 2, 3, 4, 5], 2))
```

```
[4, 5]
```

Task 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.

```
def reverse_skip(lst):
    """Extracts elements in reverse order, starting from the second-to-last element and skipping one element in between."""
    return lst[-2::-2]
print(reverse_skip([1, 2, 3, 4, 5, 6]))
```

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

Exercise on Nested Lists

Task 1: Flatten a Nested List

Write a Python function that takes a nested list and flattens it into a single list.

```
def flatten(lst):
    """Flattens a nested list into a single list."""
    flat_list = []
    for sublist in lst:
        if isinstance(sublist, list):
            flat_list.extend(flatten(sublist))
        else:
            flat_list.append(sublist)
    return flat_list

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

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

Task 2: Accessing Nested List Elements

Write a Python function that extracts a specific element from a nested list given its indices.

```
def access_nested_element(lst, indices):
    """Extracts a specific element from a nested list given its indices."""
    for index in indices:
        lst = lst[index]
    return lst

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

```
6
```

Task 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).

```
def sum_nested(lst):
    """Calculates the sum of all the numbers in a nested list."""
    total = 0
    for element in lst:
        if isinstance(element, list):
            total += sum_nested(element)
        else:
            total += element
    return total

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

```
21
```

Task 4: Remove Specific Element from a Nested List

Write a Python function that removes all occurrences of a specific element from a nested list.

```
def remove_element(lst, elem):
    """Removes all occurrences of a specific element from a nested list."""
    return [[remove_element(item, elem) if isinstance(item, list) else item for item in sublist if item != elem] for sublist in lst]

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

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

Task 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).

```
def find_max(lst):
    """Finds the maximum element in a nested list."""
    max_val = float('-inf')
    for element in lst:
        if isinstance(element, list):
            max_val = max(max_val, find_max(element))
        else:
            max_val = max(max_val, element)
    return max_val

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

```
6
```

Task 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.

```
def count_occurrences(lst, elem):
    """Counts how many times a specific element appears in a nested list."""
    count = 0
    for element in lst:
        if isinstance(element, list):
            count += count_occurrences(element, elem)
        else:
            if element == elem:
                count += 1
    return count

print(count_occurrences([[1, 2], [2, 3], [2, 4]], 2))
```

```
3
```

Task 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.

```
def deep_flatten(lst):
    """Flattens a list of lists of lists into a single list, regardless of depth."""
    flat_list = []
    for element in lst:
        if isinstance(element, list):
            flat_list.extend(deep_flatten(element))
        else:
            flat_list.append(element)
    return flat_list

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

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

Task 8: Nested List Average

Write a Python function that calculates the average of all elements in a nested list.

```
def average_nested(lst):
    """Calculates the average of all elements in a nested list."""
    total = sum_nested(lst)
    count = len(deep_flatten(lst))
    return total / count

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

```
3.5
```

Exercise on NumPy

Problem 1: Array Creation

1. Initialize an empty array with size 2×2
2. Initialize an all-one array with size 4×2.
3. Return a new array of given shape and type, filled with fill_value.
4. Return a new array of zeros with the same shape and type as a given array.
5. Return a new array of ones with the same shape and type as a given array.
6. Convert a list [1, 2, 3, 4] to a NumPy array.

```
import numpy as np

# 1. Initialize an empty array with size 2x2
empty_array = np.empty((2, 2))
print("Empty array:\n", empty_array)

# 2. Initialize an all-one array with size 4x2
ones_array = np.ones((4, 2))
print("All ones array:\n", ones_array)

# 3. Return a new array of given shape and type, filled with fill_value
fill_value_array = np.full((3, 3), 5)
print("Fill value array:\n", fill_value_array)

# 4. Return a new array of zeros with the same shape and type as a given array
given_array = np.array([[1, 2], [3, 4]])
zeros_like_array = np.zeros_like(given_array)
print("Zeros like array:\n", zeros_like_array)

# 5. Return a new array of ones with the same shape and type as a given array
ones_like_array = np.ones_like(given_array)
print("Ones like array:\n", ones_like_array)

# 6. Convert a list [1, 2, 3, 4] to a NumPy array
new_list = [1, 2, 3, 4]
numpy_array = np.array(new_list)
print("Numpy array from list:\n", numpy_array)
```

```
Empty array:
 [[1.88762543e-316 0.00000000e+000]
 [0.00000000e+000 1.63730806e-306]]
All ones array:
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
Fill value array:
 [[5 5 5]
 [5 5 5]
 [5 5 5]]
Zeros like array:
 [[0 0]
 [0 0]]
Ones like array:
 [[1 1]
 [1 1]]
Numpy array from list:
 [1 2 3 4]
```

Problem 2: Array Manipulation

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

2. Create a 3×3 matrix with values ranging from 0 to 8.
3. Create a 3×3 identity matrix.
4. Create a random array of size 30 and find the mean of the array.
5. Create a 10×10 array with random values and find the minimum and maximum values.
6. Create a zero array of size 10 and replace the 5th element with 1.
7. Reverse an array [1, 2, 0, 0, 4, 0].
8. Create a 2D array with 1 on the border and 0 inside.
9. Create an 8×8 matrix and fill it with a checkerboard pattern.

```
import numpy as np

# 1. Create an array with values ranging from 10 to 49
array_10_to_49 = np.arange(10, 50)
print("Array from 10 to 49:\n", array_10_to_49)

# 2. Create a 3x3 matrix with values ranging from 0 to 8
matrix_3x3 = np.arange(9).reshape(3, 3)
print("3x3 matrix:\n", matrix_3x3)

# 3. Create a 3x3 identity matrix
identity_matrix = np.eye(3)
print("Identity matrix:\n", identity_matrix)

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

# 5. Create a 10x10 array with random values and find the min and max
random_10x10 = np.random.random((10, 10))
min_value = random_10x10.min()
max_value = random_10x10.max()
print("Min value:\n", min_value)
print("Max value:\n", max_value)

# 6. Create a zero array of size 10 and replace the 5th element with 1
zero_array = np.zeros(10)
zero_array[4] = 1
print("Zero array with 5th element replaced:\n", zero_array)

# 7. Reverse an array [1, 2, 0, 0, 4, 0]
array_to_reverse = np.array([1, 2, 0, 0, 4, 0])
reversed_array = array_to_reverse[::-1]
print("Reversed array:\n", reversed_array)

# 8. Create a 2D array with 1 on the border and 0 inside
border_array = np.ones((5, 5))
border_array[1:-1, 1:-1] = 0
print("Border array:\n", border_array)

# 9. Create an 8x8 matrix and fill it with a checkerboard pattern
checkerboard = np.zeros((8, 8))
checkerboard[1::2, ::2] = 1
checkerboard[::2, 1::2] = 1
print("Checkerboard pattern:\n", checkerboard)
```

```
Array from 10 to 49:
 [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
3x3 matrix:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Identity matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Random array mean:
 0.5607180864969277
Min value:
 0.0054838912491053105
Max value:
 0.9804203407599374
Zero array with 5th element replaced:
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Reversed array:
  [0 4 0 0 2 1]
 Border 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.]]
 Checkerboard pattern:
  [[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

Given arrays:

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

Perform the following tasks:

1. Add the two arrays.
2. Subtract the two arrays.
3. Multiply the array with any integer 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, and x and y.
6. Concatenate x and y along rows and v and w along columns.
7. Concatenate x and v; if you get an error, observe and explain why.

```
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])

# 1. Add the two arrays
add_result = x + y
print("Addition:\n", add_result)

# 2. Subtract the two arrays
subtract_result = x - y
print("Subtraction:\n", subtract_result)

# 3. Multiply the array with any integer
multiply_result = x * 2
print("Multiplication:\n", multiply_result)

# 4. Find the square of each element of the array
square_result = np.square(x)
print("Square:\n", square_result)

# 5. Find the dot product between v and w, x and v, and x and y
dot_vw = np.dot(v, w)
dot_xv = np.dot(x, v)
dot_xy = np.dot(x, y)
print("Dot product v and w:\n", dot_vw)
print("Dot product x and v:\n", dot_xv)
print("Dot product x and y:\n", dot_xy)

# 6. Concatenate x and y along rows and v and w along columns
concat_rows = np.concatenate((x, y), axis=0)
concat_cols = np.column_stack((v, w))
print("Concatenate along rows:\n", concat_rows)
print("Concatenate along columns:\n", concat_cols)

# 7. Concatenate x and v; if you get an error, observe and explain why
try:
    concat_xv = np.concatenate((x, v))
    print("Concatenate x and v:\n", concat_xv)
except ValueError as e:
    print("Error:", e)
    print("Explanation: x is a 2D array (2x2), while v is a 1D array (2,). Their shapes are incompatible for concatenation.")
```

```
Addition:
 [[ 6  8]
 [10 13]]
Subtraction:
 [[-4 -4]
 [-4 -3]]
Multiplication:
 [[ 2  4]
 [ 6 10]]
Square:
 [[ 1  4]
 [ 9 25]]
Dot product v and w:
 219
Dot product x and v:
 [29 77]
Dot product x and y:
 [[19 22]
 [50 58]]
Concatenate along rows:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]
Concatenate along columns:
 [[ 9 11]
 [10 12]]
Error: 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 ha
Explanation: x is a 2D array (2x2), while v is a 1D array (2,). Their shapes are incompatible for concatenation.
```

Problem 4: Matrix Operations Given arrays:

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

B = np.array([[5, 3], [2, 1]])

Prove

1. Prove AA^-1 = I
2. Prove AB ≠ BA
3. Prove (AB)^T = B^T A^T

```
import numpy as np

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

# 1. Prove AA^{-1} = I
A_inv = np.linalg.inv(A)
identity = np.dot(A, A_inv)
print("AA^{-1}:\n", identity)

# 2. Prove AB != BA
AB = np.dot(A, B)
BA = np.dot(B, A)
print("AB:\n", AB)
print("BA:\n", BA)
print("AB != BA:", not np.array_equal(AB, BA))

# 3. Prove (AB)^T = B^T A^T
AB_T = AB.T
B_T_A_T = np.dot(B.T, A.T)
print("(AB)^T:\n", AB_T)
print("B^T A^T:\n", B_T_A_T)
print("(AB)^T == B^T A^T:", np.array_equal(AB_T, B_T_A_T))
```

```
AA^{-1}:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
AB:
 [[23 13]
 [51 29]]
BA:
 [[36 44]
 [13 16]]
AB != BA: True
```

```
(AB)^T:
 [[23 51]
 [13 29]]
B^T A^T:
 [[23 51]
 [13 29]]
(AB)^T == B^T A^T: True
```

Experiment: How Fast is NumPy?

Compare the performance of operations using plain Python lists and NumPy arrays.

```
import numpy as np
import time

# Element-wise Addition
size = 10**6
a_list = list(range(size))
b_list = list(range(size))
a_np = np.arange(size)
b_np = np.arange(size)

# Python Lists
start_time = time.time()
result_list = [a + b for a, b in zip(a_list, b_list)]
end_time = time.time()
print("Python Lists Addition Time:", end_time - start_time)

# NumPy Arrays
start_time = time.time()
result_np = a_np + b_np
end_time = time.time()
print("NumPy Arrays Addition Time:", end_time - start_time)

# Element-wise Multiplication
# Python Lists
start_time = time.time()
result_list = [a * b for a, b in zip(a_list, b_list)]
end_time = time.time()
print("Python Lists Multiplication Time:", end_time - start_time)

# NumPy Arrays
start_time = time.time()
result_np = a_np * b_np
end_time = time.time()
print("NumPy Arrays Multiplication Time:", end_time - start_time)

# Dot Product
# Python Lists
start_time = time.time()
dot_list = sum(a * b for a, b in zip(a_list, b_list))
end_time = time.time()
print("Python Lists Dot Product Time:", end_time - start_time)

# NumPy Arrays
start_time = time.time()
dot_np = np.dot(a_np, b_np)
end_time = time.time()
print("NumPy Arrays Dot Product Time:", end_time - start_time)
```

```
Python Lists Addition Time: 0.11620450019836426
NumPy Arrays Addition Time: 0.008435726165771484
Python Lists Multiplication Time: 0.11697578430175781
NumPy Arrays Multiplication Time: 0.011538505554199219
Python Lists Dot Product Time: 0.08814382553100586
NumPy Arrays Dot Product Time: 0.0020143985748291016
```