WORKSHEET - 0

Name: **Sanam Tamang**
UniID: **2357606**

PART 1

Part 1.1: FUNCTIONS

```
# Task 1: Python program that converts between different units of measurement

def convertLength(inputValue, inputUnit):
    if inputUnit == 'm':
        return inputValue * 3.28084
    elif inputUnit == 'ft':
        return inputValue / 3.28084
    else:
        raise ValueError("Invalid unit!")

def convertWeight(inputValue, inputUnit):
    if inputUnit == 'kg':
        return inputValue * 2.20462
    elif inputUnit == 'lbs':
        return inputValue / 2.20462
    else:
        raise ValueError("Invalid unit!")

def convertVolume(inputValue, inputUnit):
    if inputUnit == 'L':
        return inputValue * 0.264172
    elif inputUnit == 'gal':
        return inputValue / 0.264172
    else:
        raise ValueError("Invalid unit!")

def unitConverter():
    print("| UNIT CONVERTER |")
    conversionOptions = {
        '1': 'Length (m to ft)',
        '2': 'Weight (kg to lbs)',
        '3': 'Volume (L to gal)'
    }

    for optionKey, optionValue in conversionOptions.items():
        print(f"{optionKey}. {optionValue}")
    try:
        userChoice = input("Choose a conversion (1-3): ").strip()
        if userChoice == '1':
            inputUnit = input("Enter unit to convert from (m or ft): ").strip().lower()
            inputValue = float(input("Enter value: "))
            convertedValue = convertLength(inputValue, inputUnit)
            outputUnit = 'ft' if inputUnit == 'm' else 'm'

        elif userChoice == '2':
            inputUnit = input("Enter unit to convert from (kg or lbs): ").strip().lower()
            inputValue = float(input("Enter value: "))
            convertedValue = convertWeight(inputValue, inputUnit)
            outputUnit = 'lbs' if inputUnit == 'kg' else 'kg'

        elif userChoice == '3':
            inputUnit = input("Enter unit to convert from (L or gal): ").strip().lower()
            inputValue = float(input("Enter value: "))
            convertedValue = convertVolume(inputValue, inputUnit)
            outputUnit = 'gal' if inputUnit == 'L' else 'L'

        else:
            print("Please choose a option.")
            return
        print(f"Converted Value: {convertedValue:.3f} {outputUnit}")

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

unitConverter()
```

```
| UNIT CONVERTER |
1. Length (m to ft)
2. Weight (kg to lbs)
3. Volume (L to gal)
Choose a conversion (1-3): 2
Enter unit to convert from (kg or lbs): kg
Enter value: 58
Converted Value: 127.868 lbs
```

```
# TASK 2: Python program that performs various mathematical operations on a list of numbers

def computeSum(inputValues):
    return sum(inputValues)

def computeAverage(inputValues):
    if not inputValues:
        raise ValueError("Cannot calculate average of an empty list")
    return sum(inputValues) / len(inputValues)

def determineMaxValue(inputValues):
    if not inputValues:
        raise ValueError("Cannot find maximum of an empty list")
    return max(inputValues)

def determineMinValue(inputValues):
    if not inputValues:
        raise ValueError("Cannot find minimum of an empty list")
    return min(inputValues)

def matOP():
    print("\nMathematical Operations:")
    print("1. Sum")
    print("2. Average")
    print("3. Maximum")
    print("4. Minimum")
    print("5. Exit")

    try:
        operationSelection = int(input("\nEnter your choice (1-5): "))
        if operationSelection == 5:
            print("Exiting the program...")
            return

        if operationSelection < 1 or operationSelection > 4:
            print("Invalid choice. Please enter a number between 1 and 4.")
            return

        numberInput = input("Enter numbers separated by spaces: ")
        try:
            numericCollection = [float(num) for num in numberInput.split()]

            if not numericCollection:
                print("No numbers entered.")
                return

            if operationSelection == 1:
                calculatedResult = computeSum(numericCollection)
                print(f"Sum: {calculatedResult}")
            elif operationSelection == 2:
                calculatedResult = computeAverage(numericCollection)
                print(f"Average: {calculatedResult:.2f}")
            elif operationSelection == 3:
                calculatedResult = determineMaxValue(numericCollection)
                print(f"Maximum: {calculatedResult}")
            elif operationSelection == 4:
                calculatedResult = determineMinValue(numericCollection)
                print(f"Minimum: {calculatedResult}")

        except ValueError:
            print("Invalid input. Please enter valid numbers.")
    except ValueError:
        print("Please enter a valid menu choice.")

matOP()
```

```
Mathematical Operations:
1. Sum
2. Average
3. Maximum
4. Minimum
5. Exit

Enter your choice (1-5): 4
Enter numbers separated by spaces: 6 7 9 99
Minimum: 6.0
```

Part 1.2: LIST MANIPULATION

```
# This is the test list where we will perfrom manipulation
test_list = [1, 2, 3, 4, 5, 6]
```

```
# TASK 1: Extract Every Other Element
def extract_every_other(lst):
    return lst[::2]

print("Every other element:", extract_every_other(test_list))
```

```
Every other element: [1, 3, 5]
```

```
# TASK 2: Slice a sublist
def get_sublist(lst, start, end):
    return lst[start:end+1]

print("Sublist:", get_sublist(test_list, 2, 4))
```

```
Sublist: [3, 4, 5]
```

```
# TASK 3: Reverse a list using slicing
def reverse_list(lst):
    return lst[::-1]

print("Reversed list:", reverse_list(test_list))
```

```
Reversed list: [6, 5, 4, 3, 2, 1]
```

```
# TASK 4: Remove the First and Last Elements
def remove_first_last(lst):
    return lst[1:-1]

print("Remove first and last:", remove_first_last(test_list))
```

```
Remove first and last: [2, 3, 4, 5]
```

```
# TASK 5: Get the First n Elements
def get_first_n(lst, n):
    return lst[:n]

print("First 3 elements:", get_first_n(test_list, 3))
```

```
First 3 elements: [1, 2, 3]
```

```
# TASK 6: Extract Elements from the End
def get_last_n(lst, n):
    return lst[-n:]

print("Last 2 elements:", get_last_n(test_list, 3))
```

```
Last 2 elements: [4, 5, 6]
```

```
# TASK 7: Extract Elements in Reverse Order
def reverse_skip(lst):
    return lst[::-2]

print("Reverse skip:", reverse_skip(test_list))
```

```
Reverse skip: [6, 4, 2]
```

Part 1.3: NESTED LIST

```
# This is the test list where we will perfrom nested list manipulation
test_lists = [
    [[1, 2], [3, 4], [5]],
    [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
    [[1, 2], [3, [4, 5]], 6],
    [[1, 2], [3, 2], [4, 5]]
]
```

```
#TASK 1: Flatten a Nested List
def flatten(lst):
    flattened = []
    for item in lst:
        if isinstance(item, list):
            flattened.extend(flatten(item))
        else:
            flattened.append(item)
    return flattened

print("Flatten:", flatten(test_lists[0]))
```

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

```
#TASK 2:  Accessing Nested List Elements
def access_nested_element(lst, indices):
    current = lst
    for index in indices:
        current = current[index]
    return current

print("Access Nested Element:", access_nested_element(test_lists[1], [1, 2]))
```

```
Access Nested Element: 6
```

```
#TASK 3: Sum of All Elements in a Nested List
def sum_nested(lst):
    total = 0
    for item in lst:
        if isinstance(item, list):
            total += sum_nested(item)
        else:
            total += item
    return total

print("Sum Nested:", sum_nested(test_lists[2]))
```

```
Sum Nested: 21
```

```
#TASK 4: Remove Specific Element from a Nested List
def remove_element(lst, elem):
    result = []
    for item in lst:
        if isinstance(item, list):
            sublist = remove_element(item, elem)
            if sublist:  # Only add non-empty sublists
                result.append(sublist)
        elif item != elem:
            result.append(item)
    return result
```

```
print("Remove Element:", remove_element(test_lists[3], 2))
```

```
Remove Element: [[1], [3], [4, 5]]
```

```
#TASK 5: Find the Maximum Element in a Nested List
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
print("Find Max:", find_max(test_lists[2]))
```

```
Find Max: 6
```

```
#TASK 6: Count Occurrences of an Element in a Nested List
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

print("Count Occurrences:", count_occurrences(test_lists[3], 2))
```

```
Count Occurrences: 2
```

```
#TASK 7: Flatten a List of Lists of Lists
def deep_flatten(lst):
    flattened = []
    for item in lst:
        if isinstance(item, list):
            flattened.extend(deep_flatten(item))
        else:
            flattened.append(item)
    return flattened

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

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

```
#TASK 8: Nested List Average
def average_nested(lst):
    total = sum_nested(lst)
    count = len(flatten(lst))
    return total / count if count > 0 else 0

print("Average Nested:", average_nested([[1, 2], [3, 4], [5, 6]]))
```

```
Average Nested: 3.5
```

Part 2: NUMPY

```
# Importing NumPy library before doing the NUMPY TASKS
import numpy as np
```

Basic Vector and Matrix Operation with Numpy

Problem - 1: Array Creation

```
# Task 1 : Initialize an empty array with size 2X2
emptyArray = np.empty((2, 2))
print("Empty 2x2 Array:\n", emptyArray)
```

```
Empty 2x2 Array:
 [[4.1482862e-316 0.0000000e+000]
 [0.0000000e+000 4.9406565e-324]]
```

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

```
All-Ones 4x2 Array:
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
```

```
# Task 3: Create an array filled with a specific value
fillValArray = np.full((3, 3), 5)  # 3x3 array filled with 5
print("Array Filled with Value 5:\n", fillValArray)
```

```
Array Filled with Value 5:
 [[5 5 5]
 [5 5 5]
 [5 5 5]]
```

```
# Task 4: Create an array of zeros with same shape as another array
original_array = np.array([[1, 2], [3, 4]])
zerosLikeArray = np.zeros_like(original_array)
print("Zeros Array with Same Shape:\n", zerosLikeArray)
```

```
Zeros Array with Same Shape:
 [[0 0]
 [0 0]]
```

```
# Task 5: Create an array of ones with same shape as another array
onesLikeArray = np.ones_like(original_array)
print("Ones Array with Same Shape:\n", onesLikeArray)
```

```
Ones Array with Same Shape:
 [[1 1]
 [1 1]]
```

```
# Task 6: Convert an existing list to a NumPy array
new_list = [1, 2, 3, 4]
numpyArray = np.array(new_list)
print("List Converted to NumPy Array:\n", numpyArray)
```

```
List Converted to NumPy Array:
 [1 2 3 4]
```

Problem 2 - Array Manipulation: Numerical Ranges and Array indexing

```
# Task 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)
```

```
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]
```

```
# Task 2: Create a 3X3 matrix with values ranging from 0 to 8
matrix_0_to_8 = np.arange(9).reshape(3, 3)
print("3x3 Matrix from 0 to 8:\n", matrix_0_to_8)
```

```
3x3 Matrix from 0 to 8:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
# Task 3: Create a 3X3 identity matrix
identity_matrix = np.eye(3, dtype=int)
print("3x3 Identity Matrix:\n", identity_matrix)
```

```
3x3 Identity Matrix:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
```

```
# Task 4: Create a random array of size 30 and find its mean
random_array = np.random.random(30)
array_mean = random_array.mean()
print("Random Array Mean:", array_mean)
```

```
Random Array Mean: 0.5218514101381385
```

```
# Task 5: Create a 10X10 array with random values and find min/max
random_10x10 = np.random.random((10, 10))
min_val = random_10x10.min()
max_val = random_10x10.max()
print("10x10 Random Array:")
print("Minimum value:", min_val)
print("Maximum value:", max_val)
```

```
10x10 Random Array:
Minimum value: 0.0021792582088662193
Maximum value: 0.9942328157570279
```

```
# Task 6: Create a zero array of size 10 and replace 5th element with 1
zero_array = np.zeros(10, dtype=int)
zero_array[4] = 1
print("Zero Array with 5th Element as 1:\n", zero_array)
```

```
Zero Array with 5th Element as 1:
 [0 0 0 0 1 0 0 0 0 0]
```

```
# Task 7: Reverse an array
arr = np.array([1, 2, 8, 4, 0])
reversed_arr = arr[::-1]
print("Reversed Array:\n", reversed_arr)
```

```
Reversed Array:
 [0 4 8 2 1]
```

```
# Task 8: Create a 2D array with 1 on border and 0 inside
border_array = np.zeros((5, 5), dtype=int)
border_array[0, :] = 1  # Top
border_array[-1, :] = 1  # Bottom
border_array[:, 0] = 1  # Left
border_array[:, -1] = 1  # Right
print("Border 1s, Inside 0s:\n", border_array)
```

```
Border 1s, Inside 0s:
 [[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]]
```

```
# Task 9: Create an 8X8 checkerboard pattern
checkerboard = np.zeros((8, 8), dtype=int)
checkerboard[::2, ::2] = 1  # Even Rs and Cs
checkerboard[1::2, 1::2] = 1  # Odd Rs and Cs
print("8x8 Checkerboard Pattern:\n", checkerboard)
```

```
8x8 Checkerboard Pattern:
 [[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]
 [0 1 0 1 0 1 0 1]]
```

Problem - 3: Array Operations

```
# Defining 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])
```

```
# Task 1: Add the two arrays
arraySum = x + y
print("Array Addition:")
print(arraySum)
```

```
Array Addition:
[[ 6  8]
 [10 13]]
```

```
# Task 2: Subtract the two arrays
arrayDiff = x - y
print("Array Subtraction:")
print(arrayDiff)
```

```
Array Subtraction:
[[-4 -4]
 [-4 -3]]
```

```
# Task 3: Multiply the array with an integer
scalarMultX = x * 3
scalarMultY = y * 2
print("Scalar Multiplication:")
print("x * 3:\n", scalarMultX)
print("y * 2:\n", scalarMultY)
```

```
Scalar Multiplication:
x * 3:
 [[ 3  6]
 [ 9 15]]
y * 2:
 [[10 12]
 [14 16]]
```

```
# Task 4: Find the square of each element
xSquared = x ** 2
ySquared = y ** 2
print("Array Squared:")
print("x squared:\n", xSquared)
print("y squared:\n", ySquared)
```

```
Array Squared:
x squared:
 [[ 1  4]
 [ 9 25]]
y squared:
 [[25 36]
 [49 64]]
```

```
# Task 5: Dot Products

# Dot product of v and w
dotVW = np.dot(v, w)
print("Dot Products:")
print("v · w:", dotVW)

# Dot product of x and v
dotXV = np.dot(x, v)
print("x · v:", dotXV)
```

```
# Dot product of x and y
dotXY = np.dot(x, y)
print("x · y:\n", dotXY)
```

```
Dot Products:
v · w: 219
x · v: [29 77]
x · y:
 [[19 22]
 [50 58]]
```

```
# Task 6: Concatenation

# Concatenate x and y along rows vertically
rowConcat = np.concatenate((x, y), axis=0)
print("Concatenation:")
print("x and y:\n", rowConcat)

# Concatenate v and w along columns horizontally
colConcat = np.concatenate((v.reshape(-1, 1), w.reshape(-1, 1)), axis=1)
print("v and w:\n", colConcat)
```

```
Concatenation:
x and y:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]
v and w:
 [[ 9 11]
 [10 12]]
```

```
# Task 7: Attempt to concatenate x and v
try:
    concatXV = np.concatenate((x, v), axis=0)
    print("Concatenation of x and v:", concatXV)
except ValueError as e:
    print("Error:", e)

    print("x shape:", x.shape)
    print("v shape:", v.shape)
# Cannot concatenate arrays with different dimensions
```

```
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
x shape: (2, 2)
v shape: (2,)
```

Problem 4 - Matrix Operations

```
# TASK 1: Prove following using NumPy

# Defining the matrices
matrixA = np.array([[3, 4], [7, 8]])
matrixB = np.array([[5, 3], [2, 1]])



# 1.1. Prove matrixA * matrixA^-1 = I (Identity Matrix)
def proveMatrixInverse():
    # Calculating the inverse of matrix A
    try:
        inverseA = np.linalg.inv(matrixA)
        print("A Inverse:\n", inverseA)

        # Multiplying matrix A and its inverse
        multiplicationResult = np.dot(matrixA, inverseA)
        print("\nA * A^-1:\n", multiplicationResult)

        # Creating identity matrix of same size
        identityMatrix = np.eye(matrixA.shape[0])
        print("\nIdentity Matrix:\n", identityMatrix)
        # Checking if the result is close to identity matrix
        isIdentityVerified = np.allclose(multiplicationResult, identityMatrix)
        print("\nIs A * A^-1 equal to Identity Matrix?", isIdentityVerified)
    except np.linalg.LinAlgError:
        print("Matrix A is not invertible.")

proveMatrixInverse()
```

```
A Inverse:
 [[-2.    1.  ]
 [ 1.75 -0.75]]

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

Identity Matrix:
 [[1. 0.]
 [0. 1.]]

Is A * A^-1 equal to Identity Matrix? True
```

```
# 1.2.  Prove matrixA * matrixB is not equal to matrixB * matrixA
def prove():

    # Calculating matrixA * matrixB
    productAB = np.dot(matrixA, matrixB)
    print("AB:\n", productAB)

    # Calculating matrixB * matrixA
    productBA = np.dot(matrixB, matrixA)
    print("\nBA:\n", productBA)

    # Checking if they are equal
    areProductsEqual = np.array_equal(productAB, productBA)
    print("\nAre AB and BA equal?", areProductsEqual)

prove()
```

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

BA:
 [[36 44]
 [13 16]]

Are AB and BA equal? False
```

```
# 1.3. Prove (matrixA*matrixB)^T = matrixB^T * matrixA^T
def proveTranspose():

    # Calculating matrixA * matrixB
    productAB = np.dot(matrixA, matrixB)

    # Calculating (matrixA*matrixB)^T
    transposeAB = productAB.T
    print("(AB)^T:\n", transposeAB)

    # Calculating matrixB^T * matrixA^T
    transposeB = matrixB.T
    transposeA = matrixA.T
    productTransposeBTA = np.dot(transposeB, transposeA)
    print("\nB^T * A^T:\n", productTransposeBTA)

    # Checking if they are equal
    areTransposesEqual = np.array_equal(transposeAB, productTransposeBTA)
    print("\nIs (AB)^T equal to B^T * A^T?", areTransposesEqual)

proveTranspose()
```

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

B^T * A^T:
 [[23 51]
 [13 29]]
```

```
Is (AB)^T equal to B^T * A^T? True
```

```
# TASK 2: Linear Equation using Inverse Method

# Defining the coefficient matrix A
A = np.array([[2, -3, 1],
              [1, -1, 2],
              [3, 1, -1]])

# Defining the constant matrix B
B = np.array([[-1],
              [-3],
              [ 9]])

# Calculate the inverse of matrix A
A_inverse = np.linalg.inv(A)

# Calculate the solution matrix X
X = np.dot(A_inverse, B)

# Print the solution
print("Solution:")
print("x =", X[0,0])
print("y =", X[1,0])
print("z =", X[2,0])
```

```
Solution:
x = 2.0
y = 1.0
z = -2.0
```

HOW FAST IS NUMPY?

```
# Importing time for comaprison
import time
```

```
# Size of list for task 1, 2, 3
size = 1000000
```

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

```
# 1. ELement wise addition

# LIST
start_time = time.time()
result_list_add = [list1[i] + list2[i] for i in range(size)]
end_time = time.time()
print(f"List Time: {end_time - start_time:.4f} seconds")

# NUMPY ARRAYS
array1 = np.arange(size)
array2 = np.arange(size)
start_time = time.time()
result_array_add = array1 + array2
end_time = time.time()
print(f"NumPy Time: {end_time - start_time:.4f} seconds")
```

```
List Time: 0.0801 seconds
NumPy Time: 0.0022 seconds
```

```
# 2. ELement wise multiplication

# LIST
start_time = time.time()
result_list_mult = [list1[i] * list2[i] for i in range(size)]
end_time = time.time()
print(f"List Time: {end_time - start_time:.4f} seconds")
```

```
# NUMPY ARRAYS
start_time = time.time()
result_array_mult = array1 * array2
end_time = time.time()
print(f"NumPy Time: {end_time - start_time:.4f} seconds")
```

```
List Time: 0.1367 seconds
NumPy Time: 0.0024 seconds
```

```
# 3. Dot product

# LIST
start_time = time.time()
dot_product_list = sum(list1[i] * list2[i] for i in range(size))
end_time = time.time()
print(f"List Time: {end_time - start_time:.4f} seconds")

# NUMPY ARRAYS
start_time = time.time()
dot_product_array = np.dot(array1, array2)
end_time = time.time()
print(f"NumPy Time: {end_time - start_time:.4f} seconds")
```

```
List Time: 0.0891 seconds
NumPy Time: 0.0020 seconds
```

```
# FOR TASK 4
matrix_size = 1000
```

```
# 4. Matrix Multiplication

# LIST
matrix1_list = [[i + j for j in range(matrix_size)] for i in range(matrix_size)]
matrix2_list = [[i + j for j in range(matrix_size)] for i in range(matrix_size)]

def matMultList(m1, m2):
    result = [[0 for _ in range(matrix_size)] for _ in range(matrix_size)]
    for i in range(matrix_size):
        for j in range(matrix_size):
            for k in range(matrix_size):
                result[i][j] += m1[i][k] * m2[k][j]
    return result

start_time = time.time()
result_matrix_list = matMultList(matrix1_list, matrix2_list)
end_time = time.time()
print(f"List Matrix Time: {end_time - start_time:.6f} seconds")

# NUMPY Arrays
matrix1_array = np.arange(matrix_size * matrix_size).reshape(matrix_size, matrix_size)
matrix2_array = np.arange(matrix_size * matrix_size).reshape(matrix_size, matrix_size)

start_time = time.time()
result_matrix_array = np.dot(matrix1_array, matrix2_array)
end_time = time.time()
print(f"NumPy Matrix Time: {end_time - start_time:.6f} seconds")
```

```
List Matrix Time: 220.963398 seconds
NumPy Matrix Time: 1.768439 seconds
```

As we can see NUMPY arrays are quicker than lists, especially in Task 4.