2357572-suraj-kanwar-worksheet0

March 5, 2025

Student ID: 2357572
Student Name: Suraj Kanwar

1 4 TO-DO-Task

4.1 Exercise on Functions: ### Task 1: Unit Conversion Program

Create a Python program that:

1. Prompts the user to choose a conversion type (length, weight, or volume).
2. Asks for the value to convert.
3. Performs the conversion and displays the result.
4. Handles errors for invalid input or unsupported conversions.

Requirements: - Define a function to perform the conversion. - Use try-except blocks for error handling. - Include docstrings for your function.

Conversions: 1. **Length:** meters (m) to feet (ft), feet (ft) to meters (m) 2. **Weight:** kilograms (kg) to pounds (lbs), pounds (lbs) to kilograms (kg) 3. **Volume:** liters (L) to gallons (gal), gallons (gal) to liters (L)

[ ]:
```python
def unit_conversion():
    try:
        print("Choose conversion:")
        print("1. Length (m-ft / ft-m)")
        print("2. Weight (kg-lbs / lbs-kg)")
        print("3. Volume (l-gal / gal-l)")

        choice = input("Choice (1-3): ")

        if choice == "1":
            direction = input("Convert (1) m to ft or (2) ft to m: ")
            if direction not in ["1", "2"]:
                print("Invalid direction choice!")
                return

            value = float(input("Enter value to convert: "))
            if direction == "1":
                result = value * 3.28084
                print(f"{value} meters = {result:.2f} feet")
```

```
            else:
                result = value / 3.28084
                print(f"{value} feet = {result:.2f} meters")

        elif choice == "2":
            direction = input("Convert (1) kg to lbs or (2) lbs to kg: ")
            if direction not in ["1", "2"]:
                print("Invalid direction choice!")
                return

            value = float(input("Enter value to convert: "))
            if direction == "1":
                result = value * 2.20462
                print(f"{value} kilograms = {result:.2f} pounds")
            else:
                result = value / 2.20462
                print(f"{value} pounds = {result:.2f} kilograms")

        elif choice == "3":
            direction = input("Convert (1) l to gal or (2) gal to l: ")
            if direction not in ["1", "2"]:
                print("Invalid direction choice!")
                return

            value = float(input("Enter value to convert: "))
            if direction == "1":
                result = value * 0.264172
                print(f"{value} liters = {result:.2f} gallons")
            else:
                result = value / 0.264172
                print(f"{value} gallons = {result:.2f} liters")

        else:
            print("Invalid choice!")

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

unit_conversion()
```

```
Choose conversion:
1. Length (m-ft / ft-m)
2. Weight (kg-lbs / lbs-kg)
3. Volume (l-gal / gal-l)
Choice (1-3): 2
Convert (1) kg to lbs or (2) lbs to kg: 2
Enter value to convert: 44
```

```
44.0 pounds = 19.96 kilograms
```

1.0.1 Task 2: Mathematical Operations on a List

Create a Python program that:

1. Prompts the user to choose an operation (sum, average, maximum, or minimum).
2. Asks for a list of numbers (separated by spaces).
3. Performs the selected operation and displays the result.
4. Handles errors, such as non-numeric input or empty lists.

Requirements: - Define a function for each operation (sum, average, maximum, minimum). - Use try-except blocks for error handling. - Include docstrings in each function.

[ ]:
```python
def list_operations():
    try:
        print("Choose operation:")
        print("1. Sum")
        print("2. Average")
        print("3. Maximum")
        print("4. Minimum")

        choice = input("Choice (1-4): ")

        if choice not in ["1", "2", "3", "4"]:
            print("Invalid choice!")
            return

        n = list(map(float, input("Enter a list of numbers separated by spaces:␣
↪").split()))

        if not n:
            print("The list cannot be empty.")
            return

        if choice == "1":
            result = sum(n)
            print(f"The sum is: {result}")
        elif choice == "2":
            result = sum(n) / len(n)
            print(f"The average is: {result:.2f}")
        elif choice == "3":
            result = max(n)
            print(f"The maximum is: {result}")
        elif choice == "4":
            result = min(n)
            print(f"The minimum is: {result}")

    except ValueError:
```

```
        print("Invalid input. Please enter numbers separated by spaces.")

list_operations()
```

```
Choose operation:
1. Sum
2. Average
3. Maximum
4. Minimum
Choice (1-4): 2
Enter a list of numbers separated by spaces: 2 5 7 9 3
The average is: 5.20
```

2 4.2 List Manipulation Exercises

1. Extract Every Other Element:

Write a function extract_every_other(lst) that extracts every other element from a list, starting from the first element.

Example: Input: [1, 2, 3, 4, 5, 6], Output: [1, 3, 5]

[ ]:
```
def extract_every_other(lst):
    """Returns every other element from the list."""
    return lst[::2]

lst = [1, 2, 3, 4, 5, 6]
print("Every other element:", extract_every_other(lst))
```

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

2. Slice a Sublist

Write a function get_sublist(lst, start, end) that returns a sublist from index start to end (inclusive).

Example: Input: [1, 2, 3, 4, 5, 6], start=2, end=4, Output: [3, 4, 5]

[ ]:
```
def get_sublist(lst, start, end):
    """Returns a sublist from start to end index."""
    return lst[start:end+1]

lst = [1, 2, 3, 4, 5, 6]
print("Sublist from index 2 to 4:", get_sublist(lst, 2, 4))
```

```
Sublist from index 2 to 4: [3, 4, 5]
```

3. Reverse a List Using Slicing

Write a function reverse_slicing_list(lst) that reverses a list using slicing.

Example: Input: [1, 2, 3, 4, 5], Output: [5, 4, 3, 2, 1]

[ ]:
```python
def reverse_list(lst):
    """Reverses the list."""
    return lst[::-1]

lst = [1, 2, 3, 4, 5]
print("Reversed List:", reverse_list(lst))
```

```
Reversed List: [5, 4, 3, 2, 1]
```

4. Remove the First and Last Elements

Write a function remove_first_last(lst) that removes the first and last elements of a list and returns the resulting sublist.

Example: Input: [1, 2, 3, 4, 5], Output: [2, 3, 4]

[ ]:
```python
def remove_first_last(lst):
    """Removes the first and last element from the list."""
    return lst[1:-1]

lst = [1, 2, 3, 4, 5]
print("List without first and last elements:", remove_first_last(lst))
```

```
List without first and last elements: [2, 3, 4]
```

5. Get the First n Elements

Write a function get_first_n(lst, n) that returns the first n elements of a list.

Example: Input: [1, 2, 3, 4, 5], n=3, Output: [1, 2, 3]

[ ]:
```python
def get_first_n(lst, n):
    """Returns the first n elements of the list."""
    return lst[:n]

lst = [1, 2, 3, 4, 5]
print("First 3 elements:", get_first_n(lst, 3))
```

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

6. Extract Elements from the End

Write a function get_last_n(lst, n) that returns the last n elements of a list.

Example: Input: [1, 2, 3, 4, 5], n=2, Output: [4, 5]

[ ]:
```python
def get_last_n(lst, n):
    """Returns the last n elements of the list."""
    return lst[-n:]

lst = [1, 2, 3, 4, 5]
print("Last 2 elements:", get_last_n(lst, 2))
```

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

7. Extract Elements in Reverse Order

Write a function reverse_skip(lst) that returns every second element starting from the second-to-last, moving backward.

Example: Input: [1, 2, 3, 4, 5, 6], Output: [5, 3, 1]

[ ]:
```python
def reverse_skip(lst):
    """Returns elements in reverse order, skipping one in between."""
    return lst[-2::-2]

lst = [1, 2, 3, 4, 5, 6]
print("Elements in reverse order, skipping one:", reverse_skip(lst))
```

```
Elements in reverse order, skipping one: [5, 3, 1]
```

3 4.3 Exercise on Nested List

1.Flatten a Nested List:

Requirement: 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].

[ ]:
```python
def flatten(nested_list):
    """Flattens a nested list."""
    flattened_list = []
    for sublist in nested_list:
        if isinstance(sublist, list):
            flattened_list.extend(sublist)
        else:
            flattened_list.append(sublist)
    return flattened_list

nested_list = [[1, 2], [3, 4], [5]]
print("Flattened List:", flatten(nested_list))
```

```
Flattened List: [1, 2, 3, 4, 5]
```

2. Accessing Nested List Elements:

Requirement: 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.

[ ]:
```python
def access_nested_element(lst, indices):
    """Accesses an element in a nested list by a list of indices."""
```

```
    for index in indices:
        lst = lst[index]
    return lst

lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
indices = [1, 2]
print("Accessed Element:", access_nested_element(lst, indices))
```

```
Accessed Element: 6
```

3. Sum of All Elements in a Nested List:

Requirement: 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(nested_list):
    """Sums all elements in a nested list."""
    total = 0
    for item in nested_list:
        if isinstance(item, list):
            total += sum_nested(item)
        else:
            total += item
    return total

nested_list = [[1, 2], [3, [4, 5]], 6]
print("Sum of All Elements:", sum_nested(nested_list))
```

```
Sum of All Elements: 21
```

4. Remove Specific Element from a Nested List:

Requirement: 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):
    """Removes a specific element from a nested list."""
    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
print("List after removing element:", remove_element(lst, elem))
```

```
List after removing element: [[1], [3], [4, 5]]
```

5. Find the Maximum Element in a Nested List:

Requirement: 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):
    """Finds the maximum element in a nested list."""
    max_element = float('-inf')

    for sub in lst:
        if isinstance(sub, list):
            max_element = max(max_element, find_max(sub))
        else:
            max_element = max(max_element, sub)

    return max_element

lst = [[1, 2], [3, [4, 5]], 6]
print("Maximum Element:", find_max(lst))
```

```
Maximum Element: 6
```

6. Count Occurrences of an Element in a Nested List:

Requirement: 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):
    """Counts occurrences of an element in a nested list."""
    count = 0

    for item in lst:
        if isinstance(item, list):
            count += count_occurrences(item, elem)
        else:
            if item == elem:
                count += 1

    return count
```

```
lst = [[1, 2], [2, 3], [2, 4]]
elem = 2
print("Occurrences of element:", count_occurrences(lst, elem))
```

```
Occurrences of element: 3
```

7. Flatten a List of Lists of Lists:

Requirement: 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):
    """Flattens a deeply nested list."""
    result = []
    for sublist in lst:
        if isinstance(sublist, list):
            result.extend(deep_flatten(sublist))
        else:
            result.append(sublist)
    return result

lst = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
print("Deeply Flattened List:", deep_flatten(lst))
```

```
Deeply Flattened List: [1, 2, 3, 4, 5, 6, 7, 8]
```

8. Nested List Average:

Requirement: Define a function average_nested(lst) that calculates the average of all elements in a nested list.

Example: For the input [[1, 2], [3, 4], [5, 6]], the output should be 3.5.

[ ]:
```
def average_nested(lst):
    """Calculates the average of all elements in a nested list."""
    f_t = deep_flatten(lst)
    return sum(f_t) / len(f_t) if f_t else 0

lst = [[1, 2], [3, 4], [5, 6]]
print("Average of elements:", average_nested(lst))
```

```
Average of elements: 3.5
```

4 10: To-Do: NumPy Tasks

Problem 1: Array Creation

10.1 Basic Vector and Matrix Operation with Numpy.

Problem 1: Array Creation

1. Create a 2x2 empty array.

[ ]:
```
import numpy as np

arr = np.empty((2, 2))
print("Empty array (2x2):\n", arr)
```

```
Empty array (2x2):
 [[2.29195541e-316 0.00000000e+000]
 [3.66243071e-244 3.05826635e-321]]
```

2. Create a 4x2 array filled with ones.

[ ]:
```
arr = np.ones((4, 2))
print("All ones array (4x2):\n", arr)
```

```
All ones array (4x2):
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
```

3. Create an array of a given shape, filled with a specific value using np.full().

[ ]:
```
arr = np.full((3, 3), 3)
print("Array filled with 3 (3x3):\n", arr)
```

```
Array filled with 3 (3x3):
 [[3 3 3]
 [3 3 3]
 [3 3 3]]
```

4. Create a zeros array with the same shape and type as a given array using np.zeros_like().

[ ]:
```
arr = np.zeros_like(np.array([[1, 2], [3, 4]]))
print("Zeros array with same shape: \n", arr)
```

```
Zeros array with same shape:
 [[0 0]
 [0 0]]
```

5. Create an ones array with the same shape and type as a given array using np.ones_like().

[ ]:
```
arr = np.ones_like(np.array([[1, 2], [3, 4]]))
print("Ones array with same shape: \n", arr)
```

```
Ones array with same shape:
 [[1 1]
 [1 1]]
```

6. Convert the list [1, 2, 3, 4] to a NumPy array.

[ ]:
```
arr = np.array([1, 2, 3])
print("Converted list to numpy array: \n", arr)
```

```
Converted list to numpy array:
 [1 2 3]
```

Problem 2: Array Manipulation

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

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

```
Array with values 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]
```

2. Create a 3x3 matrix with values from 0 to 8.

[ ]:
```
arr = np.reshape(np.arange(9), (3, 3))
print("3x3 matrix from 0 to 8: \n", arr)
```

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

3. Create a 3x3 identity matrix.

[ ]:
```
arr = np.eye(3)
print("3x3 identity matrix: \n", arr)
```

```
3x3 identity matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

4. Create a random array of size 30 and find its mean.

[ ]:
```
arr = np.random.random(30)
print("Random array (size 30):\n", arr)
print("Mean of the random array: ", arr.mean())
```

```
Random array (size 30):
 [0.00214915 0.38881459 0.50525519 0.73776035 0.95537202 0.48801843
 0.64163712 0.48151579 0.46773385 0.62608498 0.21962938 0.98420597
 0.70964566 0.3824413  0.31663868 0.17285272 0.18805101 0.23327971
 0.08899897 0.21378361 0.79481176 0.15993018 0.30514274 0.87990563
 0.80638498 0.75638977 0.67556355 0.39295478 0.43592838 0.41528658]
Mean of the random array:  0.4808722279612431
```

5. Create a 10x10 array with random values and find its min and max.

[ ]:
```
arr = np.random.random((10, 10))
print("10x10 random array:\n", arr)
print("Min value in array: ", arr.min())
print("Max value in array: ", arr.max())
```

```
10x10 random array:
 [[0.36470752 0.52199972 0.17282803 0.70978426 0.40933281 0.07509069
  0.74566883 0.29371208 0.39569377 0.23232134]
 [0.77039818 0.6592821  0.1308961  0.49003674 0.0828346  0.88539961
  0.24691557 0.51846005 0.92597433 0.57471666]
 [0.46163102 0.65307478 0.1455611  0.08198276 0.89628224 0.58341557
  0.11145741 0.87626945 0.80365439 0.48614484]
 [0.01957532 0.04018344 0.34798237 0.69929829 0.72568544 0.14386027
  0.82533149 0.9340268  0.8867955  0.1122775 ]
 [0.88830981 0.98418005 0.95454638 0.84015888 0.47361966 0.82118217
  0.35459715 0.89414589 0.13568853 0.16193293]
 [0.26869916 0.94888198 0.32841632 0.64407271 0.9334596  0.03117297
  0.22023815 0.66835932 0.44574871 0.49538109]
 [0.9136078  0.72462135 0.48269572 0.50800264 0.65097075 0.25132504
  0.04977035 0.8092325  0.30682262 0.92627521]
 [0.12741297 0.3161371  0.88182887 0.12276609 0.85648656 0.5355928
  0.09596015 0.92022466 0.71525456 0.70824837]
 [0.14296319 0.23169552 0.42851034 0.57015662 0.54888192 0.89846157
  0.57347802 0.75583302 0.46343847 0.75530949]
 [0.54028567 0.23071327 0.78364037 0.04896779 0.86798374 0.13561071
  0.50639061 0.0304578  0.64364533 0.7495439 ]]
Min value in array:  0.019575322028411057
Max value in array:  0.9841800466625371
```

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

[ ]:
```
arr = np.zeros(10)
arr[4] = 1
print("Zero array with 5th element replaced by 1: \n", arr)
```

```
Zero array with 5th element replaced by 1:
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

7. Reverse the array [1, 2, 0, 0, 4, 0].

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

```
Reversed array:
 [0 4 0 0 2 1]
```

8. Create a 2D array with 1 on the border and 0 inside.

[ ]:
```
arr = np.ones((5, 5))
arr[1:-1, 1:-1] = 0
print("2D array with 1 on the border and 0 inside: \n", arr)
```

```
2D array with 1 on the 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.]]
```

9. Create an 8x8 checkerboard pattern.

[ ]:
```
arr = np.zeros((8, 8), dtype=int)
arr[1::2, ::2] = 1
arr[::2, 1::2] = 1
print("8x8 Checkerboard pattern: \n", arr)
```

```
8x8 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 x = np.array([[1, 2], [3, 5]]) and y = np.array([[5, 6], [7, 8]]), and v = np.array([9, 10]), w = np.array([11, 12]):

[ ]:
```
x = np.array([[1, 2], [3, 5]])
y = np.array([[5, 6], [7, 8]])
```

1. Add the two arrays x and y.

[ ]:
```
arr = x + y
print("Sum of x and y: \n", arr)
```

```
Sum of x and y:
 [[ 6  8]
 [10 13]]
```

2. Subtract y from x.

[ ]:
```
arr = x - y
print("Difference of x and y: \n", arr)
```

```
Difference of x and y:
 [[-4 -4]
 [-4 -3]]
```

3. Multiply x with a scalar.

[ ]:
```
arr = x * 2
print("x multiplied by 2: \n", arr)
```

```
x multiplied by 2:
 [[ 2  4]
 [ 6 10]]
```

4. Find the square of each element in x.

[ ]:
```
arr = np.square(x)
print("Square of each element in x: \n", arr)
```

```
Square of each element in x:
 [[ 1  4]
 [ 9 25]]
```

5. Find dot products: v.w, x.v, and x.y.

[ ]:
```
v = np.array([9, 10])
w = np.array([11, 12])
arr = np.dot(v, w)
print("Dot product of v and w: ", arr)

arr = np.dot(x, v)
print("Dot product of x and v: \n", arr)

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

```
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]]
```

6. Concatenate x and y along rows and v and w along columns.

[ ]:
```
arr = np.concatenate((x, y), axis=0)
print("Concatenation of x and y along rows: \n", arr)

arr = np.concatenate((v.reshape(-1, 1), w.reshape(-1, 1)), axis=1)
print("Concatenation of v and w along columns: \n", arr)
```

```
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]]
```

7. Concatenate x and v; explain any errors.

[ ]:
```
try:
    arr = np.concatenate((x, v), axis=1)  # This will raise an error due to␣
 ↪shape mismatch
    print("Concatenation of x and v along columns: \n", arr)
except Exception as e:
    print("Error concatenating x and v along columns: ", e)
```

```
Error concatenating x and v along columns:  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)
```

Problem 4: Matrix Operations

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

Prove A.A ^{1} = I (Identity matrix). 1. Prove AB = BA. 2. Prove (AB) = B A . 3. Solve the system of linear equations using Inverse Methods: * 2x - 3y + z = -1 * x - y + 2z = -3 * 3x + y - z = 9

[ ]:
```
import numpy as np

# Question 1: Prove A * A ¹ = I (Identity Matrix)
A = np.array([[3, 4], [7, 8]])
A_inv = np.linalg.inv(A)
I = np.dot(A, A_inv)
print("\n1. A * A ¹ = I (Identity Matrix):\n", np.round(I, 2))

# Question 2: Prove AB = BA (Check commutativity)
B = np.array([[5, 3], [2, 1]])
AB = np.dot(A, B)
BA = np.dot(B, A)
print("\n2. Matrix Multiplication Commutativity:")
print("AB = \n", AB)
print("\nBA = \n", BA)

if np.array_equal(AB, BA):
    print("\nAB = BA, so matrices A and B commute.")
else:
    print("\nAB  BA, so matrices A and B do not commute.")
```

```
# Question 3: Prove (AB)  = B  * A
AB_T = np.transpose(AB)
BT_AT = np.dot(np.transpose(B), np.transpose(A))
print("\n3. Transpose Property: (AB)  = B  * A ")
print("(AB)  = \n", AB_T)
print("\nB  * A  = \n", BT_AT)

if np.array_equal(AB_T, BT_AT):
    print("\n(AB)  = B  * A  holds true.")
else:
    print("\n(AB)   B  * A , which contradicts the property.")

# Question 4: Solve the system of linear equations using Inverse Methods
print("\n4. Solving the system of linear equations:")
print("   2x - 3y + z  = -1")
print("   x  -  y + 2z = -3")
print("   3x +  y - z  =  9")

A_eq = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])
B_eq = np.array([-1, -3, 9])

# Solution using np.linalg.solve
X = np.linalg.solve(A_eq, B_eq)
print("\nSolution using np.linalg.solve:")
print(f"x = {X[0]:.2f}, y = {X[1]:.2f}, z = {X[2]:.2f}")

# Solution using matrix inverse
A_inv_eq = np.linalg.inv(A_eq)
X_inv_method = np.dot(A_inv_eq, B_eq)
print("\nSolution using inverse method:")
print(f"x = {X_inv_method[0]:.2f}, y = {X_inv_method[1]:.2f}, z =␣
 ↪{X_inv_method[2]:.2f}")
```

```
1. A * A ¹ = I (Identity Matrix):
 [[1. 0.]
 [0. 1.]]

2. Matrix Multiplication Commutativity:
AB =
 [[23 13]
 [51 29]]

BA =
 [[36 44]
 [13 16]]
```

```
AB  BA, so matrices A and B do not commute.

3. Transpose Property: (AB)  = B  * A
(AB)  =
 [[23 51]
 [13 29]]

B  * A  =
 [[23 51]
 [13 29]]

(AB)  = B  * A  holds true.

4. Solving the system of linear equations:
   2x - 3y + z  = -1
   x  -  y + 2z = -3
   3x +  y - z  =  9

Solution using np.linalg.solve:
x = 2.00, y = 1.00, z = -2.00

Solution using inverse method:
x = 2.00, y = 1.00, z = -2.00
```

5 10.2 Experiment: How Fast is NumPy?

Compare the performance of Python lists and NumPy arrays for:

[ ]:
```python
import time

size = 1_000_000
list1, list2 = list(range(size)), list(range(size))
array1, array2 = np.arange(size), np.arange(size)
```

1. Element-wise Addition

- Add two lists (size 1,000,000) and measure the time.
- Add two NumPy arrays (size 1,000,000) and measure the time.

[ ]:
```python
start = time.time()
list_sum = [a + b for a, b in zip(list1, list2)]
end = time.time()
print("Python List Addition Time:", end - start)

start = time.time()
numpy_sum = array1 + array2
end = time.time()
```

```
print("NumPy Array Addition Time:", end - start)
```

```
Python List Addition Time: 0.060423851013183594
NumPy Array Addition Time: 0.00570225715637207
```

2. Element-wise Multiplication

- Multiply two lists (size 1,000,000) and measure the time.
- Multiply two NumPy arrays (size 1,000,000) and measure the time.

[ ]:
```
start = time.time()
list_mult = [a * b for a, b in zip(list1, list2)]
end = time.time()
print("Python List Multiplication Time:", end - start)

start = time.time()
numpy_mult = array1 * array2
end = time.time()
print("NumPy Array Multiplication Time:", end - start)
```

```
Python List Multiplication Time: 0.05883526802062988
NumPy Array Multiplication Time: 0.0038454532623291016
```

3. Dot Product

- Compute dot product using lists (size 1,000,000) and measure the time.
- Compute dot product using NumPy arrays (size 1,000,000) and measure the time.

[ ]:
```
start = time.time()
dot_product = sum(a * b for a, b in zip(list1, list2))
end = time.time()
print("Python List Dot Product Time:", end - start)

start = time.time()
numpy_dot_product = np.dot(array1, array2)
end = time.time()
print("NumPy Array Dot Product Time:", end - start)
```

```
Python List Dot Product Time: 0.07057929039001465
NumPy Array Dot Product Time: 0.0016374588012695312
```

4. Matrix Multiplication

- Multiply two 1000×1000 matrices using lists and measure the time.
- Multiply two 1000×1000 matrices using NumPy and measure the time.

[ ]:
```
matrix_size = 1000
matrix1 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]
matrix2 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]

start = time.time()
```

```
result_matrix = [[sum(a * b for a, b in zip(row, col)) for col in␣
 ↪zip(*matrix2)] for row in matrix1]
end = time.time()
print("Python List Matrix Multiplication Time:", end - start)

matrix1_np = np.random.random((matrix_size, matrix_size))
matrix2_np = np.random.random((matrix_size, matrix_size))

start = time.time()
numpy_matrix_mult = np.dot(matrix1_np, matrix2_np)
end = time.time()
print("NumPy Array Matrix Multiplication Time:", end - start)
```

```
Python List Matrix Multiplication Time: 142.72239542007446
NumPy Array Matrix Multiplication Time: 0.05358767509460449
```