Task 1. Python Programs on lists & Dictionaries?

1. Basic List Operations:

```
# Creating and accessing lists
fruits = ['apple', 'banana', 'cherry', 'date']
numbers = [1, 2, 3, 4, 5]
mixed = [1, 'hello', 3.14, True]
print("Fruits:", fruits)
print("First fruit:", fruits[0])
print("Last fruit:", fruits[-1])
print("Slice:", fruits[1:3])
# List methods
fruits.append('elderberry')
print("After append:", fruits)
fruits.insert(2, 'blueberry')
print("After insert:", fruits)
fruits.remove('banana')
print("After remove:", fruits)
popped = fruits.pop()
print("Popped:", popped)
print("After pop:", fruits)
# List operations
numbers_doubled = [x * 2 for x in numbers]
print("Doubled numbers:", numbers_doubled)
even_numbers = [x for x in range(10) if x % 2 == 0]
print("Even numbers:", even_numbers)
```

2. List Manipulation Programs

```
# 1. Find maximum and minimum in list
def find_min_max(lst):
    return min(lst), max(lst)
```

```
numbers = [45, 23, 78, 12, 90, 34]
min_val, max_val = find_min_max(numbers)
print(f"Min: {min_val}, Max: {max_val}")
# 2. Reverse a list
def reverse_list(lst):
    return lst[::-1]
print("Reversed:", reverse_list([1, 2, 3, 4, 5]))
# 3. Remove duplicates
def remove_duplicates(lst):
    return list(set(lst))
duplicate_list = [1, 2, 2, 3, 4, 4, 5]
print("Without duplicates:", remove_duplicates(duplicate_list))
# 4. Find common elements
def find_common(list1, list2):
    return list(set(list1) & set(list2))
list_a = [1, 2, 3, 4, 5]
list_b = [4, 5, 6, 7, 8]
print("Common elements:", find_common(list_a, list_b))
# 5. List sorting and searching
def sort_and_search(lst, target):
    sorted_list = sorted(lst)
    try:
        index = sorted_list.index(target)
        return sorted_list, index
    except ValueError:
        return sorted_list, -1
numbers = [64, 34, 25, 12, 22, 11, 90]
sorted_nums, index = sort_and_search(numbers, 25)
print(f"Sorted: {sorted_nums}, Index of 25: {index}")
```

3. Basic Dictionary Operations

```
# Creating dictionaries
student = {
    'name': 'John Doe',
    'age': 20,
    'grade': 'A',
    'courses': ['Math', 'Science', 'English']
}
# Accessing values
print("Name:", student['name'])
print("Age:", student.get('age'))
print("Courses:", student.get('courses', []))
# Adding and updating
student['email'] = 'john@example.com'
student['age'] = 21
print("Updated student:", student)
# Dictionary methods
print("Keys:", list(student.keys()))
print("Values:", list(student.values()))
print("Items:", list(student.items()))
# Dictionary comprehension
squares = {x: x*x for x in range(1, 6)}
print("Squares:", squares)
```

4. Dictionary Manipulation Programs

```
# 1. Merge two dictionaries
def merge_dicts(dict1, dict2):
    merged = dict1.copy()
    merged.update(dict2)
    return merged
```

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
print("Merged:", merge_dicts(dict1, dict2))
# 2. Count frequency of elements
def count_frequency(lst):
    frequency = {}
    for item in lst:
        frequency[item] = frequency.get(item, 0) + 1
    return frequency
words = ['apple', 'banana', 'apple', 'cherry', 'banana', 'apple']
print("Word frequency:", count_frequency(words))
# 3. Invert dictionary (swap keys and values)
def invert_dict(d):
    return {v: k for k, v in d.items()}
original = {'a': 1, 'b': 2, 'c': 3}
print("Inverted:", invert_dict(original))
# 4. Find key with maximum value
def max_value_key(d):
    return max(d, key=d.get)
scores = {'Alice': 85, 'Bob': 92, 'Charlie': 78, 'Diana': 95}
print("Highest scorer:", max_value_key(scores))
# 5. Filter dictionary by value
def filter_dict(d, threshold):
    return {k: v for k, v in d.items() if v > threshold}
print("Scores above 80:", filter_dict(scores, 80))
```

Task 2: Python Programs on Searching and sorting

1. Searching Algorithms

Linear Search

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1


# Example
numbers = [64, 34, 25, 12, 22, 11, 90]
target = 22
result = linear_search(numbers, target)
print(f"Linear Search: {target} found at index {result}" if result != -1
      else f"{target} not found")
```

Binary Search (Iterative)

```
def binary_search_iterative(arr, target):
    left, right = 0, len(arr) - 1
        while left <= right:
        mid = (left + right) // 2
                if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
        return -1
```

```
# Example
sorted_numbers = sorted(numbers)
result = binary_search_iterative(sorted_numbers, target)
print(f"Binary Search (Iterative): {target} found at index {result}")
```

2. Basic Sorting Algorithms

Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # If no swapping occurred, array is sorted
        if not swapped:
            break
    return arr
# Example
unsorted = [64, 34, 25, 12, 22, 11, 90]
print("Original:", unsorted)
print("Bubble Sort:", bubble_sort(unsorted.copy()))
```

Selection Sort

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
```

```
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
print("Selection Sort:", selection_sort(unsorted.copy()))
```

Insertion Sort

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
print("Insertion Sort:", insertion_sort(unsorted.copy()))
```

Task3: Python Programs on Text Handling

1. Basic String Operations

```
# Basic string creation and methods
text = "  Hello, World!  "
name = "Python Programming"
print("Original text:", repr(text))
print("Lowercase:", text.lower())
print("Uppercase:", text.upper())
print("Title case:", name.title())
print("Capitalize:", text.strip().capitalize())
print("Stripped:", repr(text.strip()))
print("Left stripped:", repr(text.lstrip()))
```

```
print("Right stripped:", repr(text.rstrip()))
print("Replaced:", text.replace("World", "Python"))
print("Starts with 'Hello':", text.strip().startswith("Hello"))
print("Ends with '!':", text.strip().endswith("!"))
print("Find 'World':", text.find("World"))
print("Count 'l':", text.count("l"))
# String formatting
name = "Alice"
age = 25
print(f"Hello, {name}! You are {age} years old.")
print("Hello, {}! You are {} years old.".format(name, age))
print("Hello, {1}! You are {0} years old.".format(age, name))
```

2. String Manipulation Programs

```
# 1. Reverse a string
def reverse_string(s):
    return s[::-1]

text = "Hello Python"
print("Reversed:", reverse_string(text))
# 2. Check palindrome
def is_palindrome(s):
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]
print("Is 'madam' palindrome?", is_palindrome("madam"))
print("Is 'A man a plan a canal Panama' palindrome?",
      is_palindrome("A man a plan a canal Panama"))
# 3. Count vowels and consonants
def count_vowels_consonants(s):
```

```
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
  for char in s:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
    return vowel_count, consonant_count
vowels, consonants = count_vowels_consonants("Hello World!")
print(f"Vowels: {vowels}, Consonants: {consonants}")
```

 # 4. Remove punctuation

```
import string
def remove_punctuation(s):
    return s.translate(str.maketrans('', '', string.punctuation))
text = "Hello, World! How's it going?"
print("Without punctuation:", remove_punctuation(text))
```

 # 5. Word count

```
def word_count(s):
    words = s.split()
    return len(words)
text = "This is a sample sentence for word counting."
print("Word count:", word_count(text))
```

Task4: Python Programs on File Handling

1. Basic File Operations

 # 1. Creating and writing to a file

```
def create_and_write_file():
    try:
```

```
        with open('sample.txt', 'w') as file:
            file.write("Hello, World!\n")
            file.write("This is a sample text file.\n")
            file.write("Python file handling is powerful!\n")
        print("File created and written successfully!")
    except IOError as e:
        print(f"Error writing to file: {e}")
create_and_write_file()

# 2. Reading from a file
def read_file(filename):
     try:
        with open(filename, 'r') as file:
            content = file.read()
            print(f"Content of {filename}:\n{content}")
    except FileNotFoundError:
        print(f"File {filename} not found!")
    except IOError as e:
        print(f"Error reading file: {e}")
read_file('sample.txt')

# 3. Reading line by line
def read_file_lines(filename):
    try:
        with open(filename, 'r') as file:
            print(f"Reading {filename} line by line:")
            for i, line in enumerate(file, 1):
                print(f"Line {i}: {line.strip()}")
    except FileNotFoundError:
```

```
        print(f"File {filename} not found!")
read_file_lines('sample.txt')


# 4. Appending to a file
def append_to_file(filename, content):
    try:
        with open(filename, 'a') as file:
            file.write(content + "\n")
        print("Content appended successfully!")
    except IOError as e:
        print(f"Error appending to file: {e}")
append_to_file('sample.txt', "This line was appended!")
read_file('sample.txt')
```

2. File Manipulation Programs

```
import os
import shutil
# 1. File existence check and properties
def file_info(filename):
    if os.path.exists(filename):
        print(f"File: {filename}")
        print(f"Size: {os.path.getsize(filename)} bytes")
        print(f"Last modified: {os.path.getmtime(filename)}")
        print(f"Absolute path: {os.path.abspath(filename)}")
    else:
        print(f"File {filename} does not exist")
file_info('sample.txt')
# 2. Copying files
def copy_file(source, destination):
    try:
```

```
        shutil.copy2(source, destination)
        print(f"File copied from {source} to {destination}")
    except FileNotFoundError:
        print("Source file not found!")
    except PermissionError:
        print("Permission denied!")
    except Exception as e:
        print(f"Error copying file: {e}")
copy_file('sample.txt', 'sample_copy.txt')
# 3. Renaming files
def rename_file(old_name, new_name):
    try:
        os.rename(old_name, new_name)
        print(f"File renamed from {old_name} to {new_name}")
    except FileNotFoundError:
        print("File not found!")
    except Exception as e:
        print(f"Error renaming file: {e}")
rename_file('sample_copy.txt', 'renamed_sample.txt')
# 4. Deleting files
def delete_file(filename):
    try:
        os.remove(filename)
        print(f"File {filename} deleted successfully!")
    except FileNotFoundError:
        print(f"File {filename} not found!")
    except PermissionError:
        print("Permission denied!")
    except Exception as e:
```

```
        print(f"Error deleting file: {e}")
delete_file('renamed_sample.txt')
```

Task5: Python Programs for calculating Mean, Mode, Median, Variance, Standard Deviation

1. Basic Statistical Functions (Without Libraries)

```
def calculate_mean(data):
    """   Calculate the arithmetic mean (average) of a dataset
          Mean = Sum of all values / Number of values   """
    if not data:
        raise ValueError("Dataset cannot be empty")
    return sum(data) / len(data)
def calculate_median(data):
    """Calculate the median (middle value) of a dataset
          For odd length: middle element
          For even length: average of two middle elements """
    if not data:
        raise ValueError("Dataset cannot be empty")
    sorted_data = sorted(data)
    n = len(sorted_data)
    mid = n // 2
    if n % 2 == 0:
        # Even number of elements
        return (sorted_data[mid - 1] + sorted_data[mid]) / 2
    else:
        # Odd number of elements
        return sorted_data[mid]
def calculate_mode(data):
    """  Calculate the mode(s) of a dataset
```

```
    Mode = Most frequently occurring value(s)
    Returns a list of modes (can be multiple) """
    if not data:
        raise ValueError("Dataset cannot be empty")
    frequency = {}
    for value in data:
        frequency[value] = frequency.get(value, 0) + 1
    max_frequency = max(frequency.values())
    modes = [key for key, value in frequency.items() if value == max_frequency]
    return modes
def calculate_variance(data, sample=True):
    """ Calculate variance of a dataset
    sample=True: sample variance (divides by n-1)
    sample=False: population variance (divides by n) """
    if not data:
        raise ValueError("Dataset cannot be empty")
    if len(data) == 1 and sample:
        raise ValueError("Sample variance requires at least 2 data points")
    mean = calculate_mean(data)
    n = len(data)
    # Calculate sum of squared differences from mean
    squared_diff = sum((x - mean) ** 2 for x in data)
    # Apply Bessel's correction for sample variance
    divisor = n - 1 if sample else n
    return squared_diff / divisor
def calculate_std_dev(data, sample=True):
    """Calculate standard deviation
    Standard deviation = Square root of variance  """
    variance = calculate_variance(data, sample)
```

```
    return variance ** 0.5
# Example usage
dataset = [4, 7, 2, 9, 5, 7, 8, 3, 7, 2]
print("Dataset:", dataset)
print("Mean:", calculate_mean(dataset))
print("Median:", calculate_median(dataset))
print("Mode:", calculate_mode(dataset))
print("Sample Variance:", calculate_variance(dataset, sample=True))
print("Population Variance:", calculate_variance(dataset, sample=False))
print("Sample Standard Deviation:", calculate_std_dev(dataset, sample=True))
print("Population Standard Deviation:", calculate_std_dev(dataset, sample=False))
```

2. Using Statistics Library (Built-in)

```
import statistics
def calculate_with_statistics_library(data):
    """ Calculate all statistical measures using Python's statistics library    """
    if not data:
        raise ValueError("Dataset cannot be empty")
    results = {
        'mean': statistics.mean(data),
        'median': statistics.median(data),
        'mode': statistics.mode(data) if len(set(data)) > 1 else data[0],
        'multimode': statistics.multimode(data),
        'sample_variance': statistics.variance(data),
        'population_variance': statistics.pvariance(data),
        'sample_std_dev': statistics.stdev(data),
        'population_std_dev': statistics.pstdev(data)
    }
```

```
    return results
# Example usage
dataset = [4, 7, 2, 9, 5, 7, 8, 3, 7, 2]
stats_results = calculate_with_statistics_library(dataset)
print("Using statistics library:")
for key, value in stats_results.items():
    print(f"{key.replace('_', ' ').title()}: {value}")
```

Task6: Python Programs for Karl Pearson Coefficient of Correlation, Rank Correlation

1. Karl Pearson's Coefficient of Correlation

```
import math
import numpy as np
from typing import List, Tuple
def pearson_correlation(x: List[float], y: List[float]) -> float:
    """ Calculate Karl Pearson's coefficient of correlation
    Formula: r = Σ((x - x̄)(y - ȳ)) / √(Σ(x - x̄)² * Σ(y - ȳ)²)
   Parameters:
    x: List of values for variable X
    y: List of values for variable Y
    Returns:
    Pearson correlation coefficient (r)
    """
    # Validation
    if len(x) != len(y):
        raise ValueError("Both lists must have the same length")
   if len(x) < 2:
        raise ValueError("At least 2 data points are required")
    n = len(x)
```

```
    # Calculate means
    mean_x = sum(x) / n
    mean_y = sum(y) / n
    # Calculate numerator and denominators
    numerator = 0
    sum_sq_x = 0
    sum_sq_y = 0
    for i in range(n):
        diff_x = x[i] - mean_x
        diff_y = y[i] - mean_y
        numerator += diff_x * diff_y
        sum_sq_x += diff_x ** 2
        sum_sq_y += diff_y ** 2
    # Handle division by zero
    if sum_sq_x == 0 or sum_sq_y == 0:
        return 0

    # Calculate correlation coefficient
    r = numerator / math.sqrt(sum_sq_x * sum_sq_y)
    return r
def pearson_correlation_using_covariance(x: List[float], y: List[float]) -> float:
    """ Alternative method using covariance and standard deviations
       Formula: r = cov(X,Y) / (σ_x * σ_y)  """
    n = len(x)
    # Calculate means
    mean_x = sum(x) / n
    mean_y = sum(y) / n
   # Calculate covariance
    covariance = sum((x[i] - mean_x) * (y[i] - mean_y) for i in range(n)) / n
```

```
    # Calculate standard deviations
    std_x = math.sqrt(sum((xi - mean_x) ** 2 for xi in x) / n)
    std_y = math.sqrt(sum((yi - mean_y) ** 2 for yi in y) / n)
    # Handle division by zero
    if std_x == 0 or std_y == 0:
        return 0
    # Calculate correlation coefficient
    r = covariance / (std_x * std_y)
    return r
def interpret_correlation(r: float) -> str:
    """ Interpret the correlation coefficient value   """
    abs_r = abs(r)
    if abs_r == 0:
        return "No correlation"
    elif abs_r < 0.3:
        return "Weak correlation"
    elif abs_r < 0.7:
        return "Moderate correlation"
    elif abs_r < 0.9:
        return "Strong correlation"
    else:
        return "Very strong correlation"

def correlation_direction(r: float) -> str:
    """ Determine the direction of correlation   """
    if r > 0:
        return "Positive correlation"
    elif r < 0:
        return "Negative correlation"
```

```
    else:
        return "No correlation"

# Example usage
def test_pearson_correlation():
    # Example datasets
    # Perfect positive correlation
    x1 = [1, 2, 3, 4, 5]
    y1 = [2, 4, 6, 8, 10]

    # Perfect negative correlation
    x2 = [1, 2, 3, 4, 5]
    y2 = [10, 8, 6, 4, 2]

    # No correlation
    x3 = [1, 2, 3, 4, 5]
    y3 = [5, 2, 8, 1, 9]

    # Real-world example: Study hours vs Exam scores
    study_hours = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
    exam_scores = [50, 55, 65, 70, 75, 80, 85, 88, 92, 95]
   datasets = [
        ("Perfect Positive", x1, y1),
        ("Perfect Negative", x2, y2),
        ("No Correlation", x3, y3),
        ("Study vs Scores", study_hours, exam_scores)
    ]

    print("Karl Pearson's Correlation Coefficient")
```

```
    print("=" * 50)
    for name, x, y in datasets:
        r = pearson_correlation(x, y)
        r_alt = pearson_correlation_using_covariance(x, y)
        print(f"\n{name}:")
        print(f"  Method 1 (direct): r = {r:.6f}")
        print(f"  Method 2 (cov):    r = {r_alt:.6f}")
        print(f"  Interpretation:    {interpret_correlation(r)}")
        print(f"  Direction:         {correlation_direction(r)}")
test_pearson_correlation()
```

2. Spearman's Rank Correlation Coefficient

```
def assign_ranks(data: List[float]) -> List[float]:
    """Assign ranks to data, handling ties appropriately
    Parameters:
    data: List of values to rank
    Returns:
    List of ranks (with average ranks for ties)
    """
    # Create list of (value, original index)
    indexed_data = [(value, i) for i, value in enumerate(data)]
   # Sort by value
    sorted_data = sorted(indexed_data, key=lambda x: x[0])
    # Initialize ranks
    ranks = [0] * len(data)
    n = len(data)
    i = 0
    while i < n:
        # Find all elements with the same value (ties)
```

```
        j = i
        while j < n - 1 and sorted_data[j][0] == sorted_data[j + 1][0]:
            j += 1
        # Calculate average rank for this group of ties
        avg_rank = (i + j + 2) / 2  # +2 because ranks start at 1, not 0
        # Assign average rank to all tied elements
        for k in range(i, j + 1):
            original_index = sorted_data[k][1]
            ranks[original_index] = avg_rank
        i = j + 1
    return ranks
def spearman_rank_correlation(x: List[float], y: List[float]) -> float:
    """ Calculate Spearman's rank correlation coefficient
    Formula: ρ = 1 - (6 * Σd²) / (n(n² - 1))
    where d = difference in ranks
    Parameters:
    x: List of values for variable X
    y: List of values for variable Y
    Returns:
    Spearman's rank correlation coefficient (ρ)
    """
    # Validation
    if len(x) != len(y):
        raise ValueError("Both lists must have the same length")
    if len(x) < 2:
        raise ValueError("At least 2 data points are required")
    n = len(x)
    # Assign ranks
    ranks_x = assign_ranks(x)
```

```
    ranks_y = assign_ranks(y)
    # Calculate differences in ranks and squared differences
    d_squared_sum = 0
    for i in range(n):
        d = ranks_x[i] - ranks_y[i]
        d_squared_sum += d ** 2
     # Calculate Spearman's coefficient
    if all(rx == ry for rx, ry in zip(ranks_x, ranks_y)):  # Perfect correlation
        return 1.0 if ranks_x[0] == ranks_y[0] else -1.0
     rho = 1 - (6 * d_squared_sum) / (n * (n ** 2 - 1))
     return rho
def spearman_rank_correlation_pearson_method(x: List[float], y: List[float]) -> float:
    """ Alternative method: Calculate Pearson correlation on ranks
        This method handles ties better than the standard formula    """
    ranks_x = assign_ranks(x)
    ranks_y = assign_ranks(y)
    return pearson_correlation(ranks_x, ranks_y)
# Example usage
def test_spearman_correlation():
    # Example datasets
    # Perfect positive rank correlation
    x1 = [10, 20, 30, 40, 50]  # Ranks: 1,2,3,4,5
    y1 = [1, 2, 3, 4, 5]       # Ranks: 1,2,3,4,5
        # Perfect negative rank correlation
    x2 = [10, 20, 30, 40, 50]  # Ranks: 1,2,3,4,5
    y2 = [5, 4, 3, 2, 1]       # Ranks: 5,4,3,2,1
     # With ties
    x3 = [10, 20, 20, 40, 50]  # Ranks: 1, 2.5, 2.5, 4, 5
    y3 = [1, 3, 2, 4, 5]       # Ranks: 1,3,2,4,5
```

```
    # Real-world example: Student rankings in two subjects
    math_scores = [85, 92, 78, 90, 88, 95, 82]
    physics_scores = [80, 95, 75, 88, 85, 92, 78]
    datasets = [
        ("Perfect Positive", x1, y1),
        ("Perfect Negative", x2, y2),
        ("With Ties", x3, y3),
        ("Math vs Physics", math_scores, physics_scores)
    ]
    print("\n" + "=" * 50)
    print("Spearman's Rank Correlation Coefficient")
    print("=" * 50)
    for name, x, y in datasets:
        rho = spearman_rank_correlation(x, y)
        rho_alt = spearman_rank_correlation_pearson_method(x, y)

        print(f"\n{name}:")
        print(f"  Standard formula: ρ = {rho:.6f}")
        print(f"  Pearson on ranks: ρ = {rho_alt:.6f}")
        print(f"  Interpretation:   {interpret_correlation(rho)}")
        print(f"  Direction:        {correlation_direction(rho)}")
         # Show ranks for smaller datasets
        if len(x) <= 7:
            ranks_x = assign_ranks(x)
            ranks_y = assign_ranks(y)
            print(f"  Ranks X: {ranks_x}")
            print(f"  Ranks Y: {ranks_y}")
test_spearman_correlation()
```

Task 7: Python Programs on NumPy Arrays and Linear Algebra with NumPy

1. Basic NumPy Array Operations

```
import numpy as np


# 1. Creating NumPy Arrays
print("1. CREATING NUMPY ARRAYS")
print("=" * 50)


# From Python list
arr_from_list = np.array([1, 2, 3, 4, 5])
print("From list:", arr_from_list)


# Arrays with zeros
zeros_arr = np.zeros(5)
print("Zeros array:", zeros_arr)


# Arrays with ones
ones_arr = np.ones(5)
print("Ones array:", ones_arr)


# Arrays with a range
range_arr = np.arange(10)
print("Range array:", range_arr)


# Arrays with specific values
full_arr = np.full(5, 7)
print("Full array:", full_arr)
```

```
# Identity matrix
identity_mat = np.eye(3)
print("Identity matrix:\n", identity_mat)

# Random arrays
random_arr = np.random.rand(5)
print("Random array:", random_arr)

# 2. Array Properties
print("\n2. ARRAY PROPERTIES")
print("=" * 50)

arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("2D array:\n", arr_2d)
print("Shape:", arr_2d.shape)
print("Dimensions:", arr_2d.ndim)
print("Size:", arr_2d.size)
print("Data type:", arr_2d.dtype)

# 3. Array Manipulation
print("\n3. ARRAY MANIPULATION")
print("=" * 50)

# Reshaping
arr = np.arange(12)
reshaped = arr.reshape(3, 4)
print("Original:", arr)
print("Reshaped (3x4):\n", reshaped)
```

```
# Flattening
flattened = reshaped.flatten()
print("Flattened:", flattened)

# Transposing
transposed = reshaped.T
print("Transposed:\n", transposed)

# Stacking arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
stacked = np.vstack([arr1, arr2])
print("Vertical stack:\n", stacked)

# 4. Array Indexing and Slicing
print("\n4. ARRAY INDEXING AND SLICING")
print("=" * 50)

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("Original array:\n", arr)

print("Element at [1, 2]:", arr[1, 2])
print("First row:", arr[0, :])
print("Second column:", arr[:, 1])
print("Subarray (first 2 rows, last 2 columns):\n", arr[:2, -2:])

# Boolean indexing
bool_idx = arr > 5
print("Elements greater than 5:", arr[bool_idx])
```

2. NumPy Array Operations and Functions

```
import numpy as np


print("5. ARRAY OPERATIONS")
print("=" * 50)

# Basic arithmetic operations
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])

print("Array a:", a)
print("Array b:", b)
print("a + b:", a + b)
print("a - b:", a - b)
print("a * b:", a * b)
print("a / b:", a / b)
print("a ** 2:", a ** 2)

# Universal functions (ufuncs)
print("\nUniversal Functions:")
print("sin(a):", np.sin(a))
print("exp(a):", np.exp(a))
print("sqrt(a):", np.sqrt(a))
print("log(a):", np.log(a + 1))  # +1 to avoid log(0)

# Aggregation functions
print("\nAggregation Functions:")
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Array:\n", arr)
```

```
print("Sum:", np.sum(arr))
print("Sum along axis 0 (columns):", np.sum(arr, axis=0))
print("Sum along axis 1 (rows):", np.sum(arr, axis=1))
print("Mean:", np.mean(arr))
print("Max:", np.max(arr))
print("Min:", np.min(arr))
print("Standard deviation:", np.std(arr))

# Broadcasting
print("\nBroadcasting:")
arr = np.array([[1, 2, 3], [4, 5, 6]])
scalar = 2
print("Array:\n", arr)
print("Array + scalar:\n", arr + scalar)
print("Array * scalar:\n", arr * scalar)

# Vector operations
print("\nVector Operations:")
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])
print("Dot product:", np.dot(v1, v2))
print("Cross product:", np.cross(v1, v2))
```

3. Linear Algebra with NumPy

```
import numpy as np

print("6. LINEAR ALGEBRA WITH NUMPY")
print("=" * 50)
```

```
# 1. Matrix Operations
print("1. MATRIX OPERATIONS")
print("-" * 30)

# Creating matrices
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

print("Matrix A:\n", A)
print("Matrix B:\n", B)

# Matrix multiplication
print("A × B:\n", np.dot(A, B))
print("A × B (using @ operator):\n", A @ B)

# Element-wise multiplication
print("Element-wise multiplication:\n", A * B)

# Matrix transpose
print("Transpose of A:\n", A.T)

# Matrix inverse
print("Inverse of A:\n", np.linalg.inv(A))

# Matrix determinant
print("Determinant of A:", np.linalg.det(A))

# 2. Solving Linear Equations
print("\n2. SOLVING LINEAR EQUATIONS")
```

```
print("-" * 30)

# Example: Solve 3x + y = 9, x + 2y = 8
coefficients = np.array([[3, 1], [1, 2]])
constants = np.array([9, 8])

solution = np.linalg.solve(coefficients, constants)
print("Coefficient matrix:\n", coefficients)
print("Constants vector:", constants)
print("Solution (x, y):", solution)

# Verify solution
print("Verification:", coefficients @ solution)

# 3. Eigenvalues and Eigenvectors
print("\n3. EIGENVALUES AND EIGENVECTORS")
print("-" * 30)

matrix = np.array([[4, 2], [1, 3]])
eigenvalues, eigenvectors = np.linalg.eig(matrix)

print("Matrix:\n", matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)

# Verify: A × v = λ × v
for i in range(len(eigenvalues)):
    v = eigenvectors[:, i]
    λ = eigenvalues[i]
```

```
    print(f"Verification for λ={λ}: A×v = {matrix @ v}, λ×v = {λ * v}")

# 4. Matrix Decomposition
print("\n4. MATRIX DECOMPOSITION")
print("-" * 30)

# Singular Value Decomposition (SVD)
matrix = np.array([[1, 2], [3, 4], [5, 6]])
U, S, Vt = np.linalg.svd(matrix)

print("Original matrix:\n", matrix)
print("U matrix:\n", U)
print("Singular values:", S)
print("V transpose matrix:\n", Vt)

# Reconstruct matrix
reconstructed = U @ np.diag(S) @ Vt
print("Reconstructed matrix:\n", reconstructed)

# QR Decomposition
Q, R = np.linalg.qr(matrix)
print("Q matrix:\n", Q)
print("R matrix:\n", R)
print("Q × R:\n", Q @ R)
```