Lab 04: Sets and NumPy

Objectives

- To understand and apply Python sets for storing and manipulating unique data.
- To perform set operations like union, intersection, difference, and membership tests.
- To understand the NumPy library, including arrays, indexing, slicing, and basic operations.
- To practice using NumPy functions for data processing, statistical calculations, and array manipulation.

1. Python Sets

- A set in Python is an unordered collection of unique elements.
- Sets are useful for eliminating duplicates and performing mathematical set operations.

Key characteristics:

- No duplicate elements
- Unordered: elements do not have indices
- Mutable: you can add or remove elements
- Defined using curly braces {} or the set() function

In [53]:
```
# Creating Sets
# Using curly braces
fruits = {"apple", "banana", "cherry"}
print(fruits)

# Using set() function
numbers = set([1, 2, 3, 4, 5])
print(numbers)

# Empty set
empty_set = set()
print(empty_set)
```

```
{'cherry', 'banana', 'apple'}
{1, 2, 3, 4, 5}
set()
```

Set Operations

- union() or | → Combines elements from two sets
- intersection() or & → Elements common in both sets
- difference() or - → Elements in one set but not in another
- symmetric_difference() or ^ → Elements in either set but not in both
- add() → Add a single element
- remove() or discard() → Remove an element
- in → Membership test

In [72]:
```
# Examples
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Union
print(A | B)  # {1, 2, 3, 4, 5, 6}
```

```
{1, 2, 3, 4, 5, 6}
```

In [55]:
```
# Intersection
print(A & B)  # {3, 4}
```

```
{3, 4}
```

In [56]:
```
# Difference
print(A - B)  # {1, 2}
```

```
{1, 2}
```

In [57]:
```
# Symmetric Difference
print(A ^ B)  # {1, 2, 5, 6}
```

```
{1, 2, 5, 6}
```

In [58]:
```
# Membership test
print(3 in A)  # True
```

```
True
```

In [73]:
```
# Adding elements
A.add(12)
print(A)  # {1, 2, 3, 4, 10}
```

```
{1, 2, 3, 4, 12}
```

In [77]:
```
# Removing elements
A.remove(2)
print(A)  # {1, 3, 4, 10}
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-77-244ce1aed17b> in <module>
      1 # Removing elements
----> 2 A.remove(2)
      3 print(A)  # {1, 3, 4, 10}

KeyError: 2
```

Exercise 1

Create two sets of numbers. Perform union, intersection, difference, and symmetric difference operations. Test whether a number exists in the set.

Exercise 2

Write a Python program to remove duplicates from a list using sets.

2. NumPy (Numerical Python)

Background / Theory

- NumPy is a Python library for numerical computation, offering:
- Multi-dimensional arrays (ndarray)
- Mathematical, logical, and statistical operations
- Efficient data storage and computation

In [ ]:
```
# Installation
!pip install numpy
```

Python Libraries

A Python library is a collection of pre-written code that you can use to perform common tasks without having to write the code from scratch.

- Libraries help you save time and increase efficiency.
- They can include functions, classes, and modules that solve specific problems.

Think of a library in real life:

- A library has books on different topics.
- You can borrow a book when you need knowledge.
- Similarly, a Python library has pre-written code you can use.

In [78]:
```
# Importing NumPy
import numpy as np
```

NumPy Arrays

1. One-dimensional: Vector
2. Two-dimensional: Matrix
3. Multi-dimensional: Tensor

In [81]:
```
# Creating Arrays
# 1D array
# Create a 1D array from a list
arr1 = np.array([10, 20, 30, 40, 50])
print("1D Array from list:", arr1)
```

```
1D Array from list: [10 20 30 40 50]
```

In [86]:
```
# Create a 1D array with numbers from 0 to 9
arr2 = np.arange(0, 10, 1)
# start=0, stop=10 (exclusive), step=1
print("1D Array using arange:", arr2)
```

```
1D Array using arange: [0 1 2 3 4 5 6 7 8 9]
```

In [87]:
```
# Array of zeros
zeros = np.zeros(5)
print("1D Zeros array:", zeros)
```

```
1D Zeros array: [0. 0. 0. 0. 0.]
```

In [89]:
```
# Array of ones
ones = np.ones(5)
print("1D Ones array:", ones)
```

```
1D Ones array: [5. 5. 5. 5. 5.]
```

In [90]:
```
# 2D array
arr2 = np.array([[1, 2, 3],
```

```
                [4, 5, 6]])
print("2D array of shape:", arr2.shape, "elements", arr2)
```

```
2D array of shape: (2, 3) elements [[1 2 3]
 [4 5 6]]
```

In [91]:
```
# 2D array (3x3 matrix) from a nested list
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])
print("2D Array from nested list:\n", matrix1)
```

```
2D Array from nested list:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [92]:
```
# Example: Array of zeros or ones
# 2D array of zeros (2x4)
zeros_matrix = np.zeros((2, 4))
print("2D Zeros array:\n", zeros_matrix)
```

```
2D Zeros array:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [93]:
```
# 2D array of ones (3x3)
ones_matrix = np.ones((3, 3))
print("2D Ones array:\n", ones_matrix)
```

```
2D Ones array:
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

In [94]:
```
# Example: Using np.arange() and reshape()
# Create 1D array and reshape to 2D
arr = np.arange(1, 13)  # 1D array with 12 elements
matrix2 = arr.reshape(3, 4)  # reshape to 3 rows and 4 columns
print("2D Array using arange and reshape:\n", matrix2)
```

```
2D Array using arange and reshape:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

In [96]:
```
# Array with evenly spaced numbers
linspace_arr = np.linspace(0, 1, 3)
# 5 points from 0 to 1
print(linspace_arr)
```

```
[0.  0.5 1. ]
```

In [97]:
```
# Example: Random 2D array
# 2x3 array with random numbers between 0 and 1
random_matrix = np.random.rand(2, 3)
print("Random 2D array:\n", random_matrix)
```

```
Random 2D array:
 [[0.07516069 0.87831515 0.40303285]
 [0.91527841 0.25115313 0.4957104 ]]
```

In [98]:
```
# Use shape to check the dimensions:

print(matrix2.shape)  # (3, 4)

# Use dtype to check data type:
print(arr1.dtype)
```

```
(3, 4)
int32
```

Array Operations

- Element-wise addition, subtraction, multiplication, division
- Aggregation functions: sum(), mean(), max(), min(), std(), var()
- Indexing and slicing

In [ ]:
```
# Examples
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
```

In [ ]:
```
# Element-wise operations
print(a + b)  # [5 7 9]
print(a * b)  # [4 10 18]
```

In [ ]:
```
# Aggregation
print(np.sum(a))  # 6
print(np.mean(b))  # 5.0
```

In [ ]:
```
# Indexing and slicing
arr = np.array([10, 20, 30, 40, 50])
print(arr[1:4])  # [20 30 40]
```

In [101…
```
# 2D Array Operations
matrix = np.array([[1,2,3],
                   [4,5,6],
```

```
                    [7,8,9]])

# Access element at row 1, column 2
print(matrix[2,2])  # 6
```

```
9
```

In [102…

```
# Slicing rows and columns
print(matrix[:,1])  # Second column
print(matrix[1,:])  # Second row
```

```
[2 5 8]
[4 5 6]
```

In [103…

```
# Matrix addition
mat2 = np.ones((3,3))
print(matrix + mat2)
```

```
[[ 2.  3.  4.]
 [ 5.  6.  7.]
 [ 8.  9. 10.]]
```

Exercise 3

- Create a 1D array of numbers from 1 to 10.
- Print all elements greater than 5.

Exercise 4

- Create a 2D matrix and calculate sum, mean, maximum, and minimum of the matrix.

Exercise 5

- Use linspace to generate 10 numbers between 0 and 5.
- Multiply all elements by 3.

Tasks for Submission

1. Create two Python sets and demonstrate all basic set operations.
2. Write a Python program to remove duplicates from a list using sets.
3. Create a 1D NumPy array and perform element-wise arithmetic operations.
4. Write a program to find the sum, mean, and standard deviation of a NumPy array.
5. Generate a 2D array of size 3x3, slice the second row and column, and display them.
6. Use arange to create an array from 10 to 50 with step 5.
7. Use linspace to generate 15 numbers from 0 to 10 and reshape it into a 3x5 matrix.
8. Write a program to find all even numbers in a NumPy array.
9. Merge two NumPy arrays vertically and horizontally.
10. Write a program using NumPy to create a 5x5 identity matrix.