ARRAY

An array is a fundamental data structure used in programming to store multiple values of the same data type under a single variable name. Each value in an array is accessed using an index (position number). Arrays help programmers organize data efficiently and reduce the need to create many separate variables.An array is a collection of elements of the same data type stored in contiguous memory locations, and each element can be accessed using an index.

CONTINUATION

- Example (conceptual):
- Instead of writing:score1, score2, score3, score4
- We write:scores[ ]

Characteristics of Arrays

- All elements must be of the same data type (e.g., all integers).
- Stored in contiguous memory locations.
- Each element has a unique index.
- Indexing usually starts from 0 in most programming languages.
- Arrays have a fixed size once declared (in many languages).

Types of Arrays One-Dimensional Array (1D Array)

- A one-dimensional array stores elements in a **single row or list**.
- **Example: int age[5] = {10, 12, 14, 16, 18};**
  Representation:

Index	**0**	**1**	**2**	**3**	**4**
Value	**10**	**12**	**14**	**16**	**18**

Two-Dimensional Array

Two-Dimensional Array (2D Array)A two-dimensional array stores data in rows and columns (like a table or matrix).Example:

```
int marks[2][3] = {
  {70, 80, 90},
  {60, 75, 85}
};
```

Multi-Dimensional Array

Representation:

	Col 0	Col 1	Col 2
Row 0	70	80	90
Row 1	60	75	85

Multi-Dimensional Array :Arrays with more than two dimensions, commonly used in advanced applications like scientific computing and image processing.

Declaration of Arrays

The general format for declaring an array is:

DataType ArrayName[Size]; e.g

```
int numbers[10];
float prices[5];
char name[20];
```

Array operations

1.Traversing an Array

Accessing each element of the array one by one.

Example (Python):

```
arr = [10, 20, 30, 40]
for i in arr:
    print(i)
```

Use: Displaying or processing all elements.

2. Insertion

Adding a new element to the array at a specific position. Insertion can be done at the beginning, At the end, At a specific index

CONTN

Example:

```
arr = [10, 20, 30]
arr.insert(1, 15)   # Insert at index 1print(arr)
```

Deletion

3. Deletion

Removing an element from the array.

Example:

arr = [10, 20, 30, 40]

arr.remove(30)print(arr)

CONTN

Searching

Finding the position of a particular element.

(a) Linear Search

```
arr = [5, 7, 9, 11]
key = 9
for i in range(len(arr)):
    if arr[i] == key:
        print("Found at index", i)
```

CONTN

5. Sorting

Arranging elements in ascending or descending order.

Example:

```
arr = [4, 2, 9, 1]
arr.sort()print(arr)
```

6. Merging

Combining two arrays into one.

Example:

```
arr1 = [1, 2, 3]
arr2 = [4, 5]
arr3 = arr1 + arr2print(arr3)
```

STRING

A string is a data type used to store a sequence of characters such as letters, digits, symbols, and spaces. Strings are commonly used to store names, words, sentences, and text data in computer programs.A string is an array or collection of characters treated as a single data item. Examples of strings: "Hello", "Computer Science", "A123", "My name is Ismail"

Characteristics of Strings

- A string is made up of characters.
- Characters are stored in contiguous memory locations.
- Strings have a length (number of characters).
- In many languages, strings are arrays of characters.
- Strings usually end with a null character \0 (in languages like C

- . Example: "DATA"

Index	0	1	2	3
Character	D	A	T	A

Declaration of Strings

Declaration of Strings

1.In C Language

```
char name[10];
char course[] = "Computer";
```

2.In Python

```
name = "Computer Science"
```

In Java

```
String name = "Technology";
```

Accessing String Characters

Accessing String Characters

Characters in a string are accessed using index values.

name[0] // First character

name[3] // Fourth character

Common String Operations

Common String Operations

1.String Length

Finding the number of characters in a string.

C: strlen()

Python: len()

Java: length()

example : Python: len("Hello") → 5

2.String Concatenation

Joining two or more strings together.

C: strcat(str1, str2)

Python: str1 + str2

Java: str1.concat(str2)

example :"Hello" + " World" → "Hello World"

String Searching

.String Searching

Find the position of a character or substring.

C: strchr() or strstr()

Python: str.find()

Java: indexOf()

Example:"Computer".find("p") → 3

4, String Slicing / Substring

Extract part of a string.

Python: str[start:end]

Java: str.substring(start, end)

Example:

"Programming"[0:6] → "Progra"

Removing White Spaces

Trim spaces at the start or end of a string.

Python: str.strip()

Java: str.trim()

Searching

Searching

Searching is the process of finding whether a particular element exists in a data structure and, if it exists, determining its position.

Types of Searching

Linear Search (Sequential Search)

The simplest search method.

Checks each element one by one until the target is found or the list ends.

- Works on unsorted or sorted data

Algorithm for searching

Algorithm (Pseudo-code):

```
for i = 0 to n-1:
    if arr[i] == key:
        return i  // element found
return -1  // element not found
```

Example: Search for 7 in [3, 5, 7, 9] → Found at index 2

```
arr = [3, 5, 7, 9]
key = 7
for i in range(len(arr)):
    if arr[i] == key:
        print("7 found at index", i)
        break
else:
    print("7 not found")
```

Pros: Simple to implement.

Cons: Slow for large lists.

Binary Search

Binary Search

- Works only on sorted lists.
- Repeatedly divides the list in half and checks the middle element.

Algorithm (Pseudo-code):

```
low = 0
high = n-1
while low <= high:
    mid = (low + high) / 2
    if arr[mid] == key:
        return mid
    else if arr[mid] < key:
        low = mid + 1
    else:
        high = mid - 1
return -1
```

EXAMPLE

Example: Search for 7 in [2, 4, 6, 7, 9] → Found at index 3

```
arr = [2, 4, 6, 7, 9]
key = 7
low = 0
high = len(arr) - 1
while low <= high:
    mid = (low + high) // 2
    if arr[mid] == key:
        print("7 is found at index", mid)
        break
    elif arr[mid] < key:
        low = mid + 1
    else:
        high = mid - 1
else:
    print("7 is not found")
```

Output

```
7 is found at index 3
```

PROS AND CONS OF BINARY SEARCH

- Pros: Much faster than linear search for large lists.
- Cons: List must be sorted.

Sorting:

Sorting: Sorting is the process of arranging elements in a list in ascending (smallest to largest) or descending (largest to smallest) order.

Types of Sorting Algorithms

1 Bubble Sort

This is a simple comparison based sorting algorithm used to arrange elements in a List or array in ascending or Descending order. It works by repeatedly comparing adjacent element and swapping them if they are in the wrong order. It is called burble sort because larger element bubble up to the end of the list after each pass. Process continues until the list is sorted.

Pros:

- Easy to understand
- Good for small Dataset

Cons:

- Slow for large Datasets
- Not efficient compared to other sorting Algorithms

Selection Sort

Finds the smallest element and swaps it with the first element.Then finds the second smallest and swaps with the second element, and so on. The idea is to repeatedly select the smallest (or largest) element from the unsorted part of the array and place it in correct position.

Pros:

- Simple and easy to implement.
- Performs faster swaps than burble sor

Cons:

- Not efficient for large Datasets
- Not suitable for real time Applicattion

Insertion Sort

Builds a sorted list one element at a time. Inserts each new element into its correct position in the already sorted part.

Pros:

- Efficient for small Datasets and nearly sorted Datasets
- Simple and easy to understand.

Cons:

- Slower for large random lists.
- Inefficient for large Dataset.

SIMPLE RECURSIVE ALGORITHMS IN OOP

Recursion is a programming technique where a function calls itself to solve a problem. It is widely used in OOP languages like Java, Python, C++, etc.

Key idea: Solve a large problem by breaking it into smaller similar subproblems.

Characteristics of Recursive Algorithms

- Base Case: The stopping condition to prevent infinite recursion.
- Recursive Case: Part of the function that calls itself.

Must eventually reach the base case.

Examples of Simple Recursive Algorithms in OOP

Example 1: Factorial of a Number

1. Factorial of a Number

Factorial of n = n × (n-1) × (n-2) × ... × 1

```
def factorial(n):
    if n == 0:        # base case
        return 1
    else:
        return n * factorial(n - 1)
print(factorial(5))
```

2.Countdown Example

Simple recursion example.

```
def countdown(n):
    if n == 0:
        print("Done")
    else:
        print(n)
       countdown(n - 1)

countdown(5)
```