



Created By: Adithya Bandara

# SOFTWARE DEVELOPMENT SHORT NOTE

*Note: This note was generated using ChatGPT, Google, and lecture notes. Please engage in independent study for further understanding.*

## What is a program?

A program is a set of instructions that tells a computer what to do.

## What is a computer software?

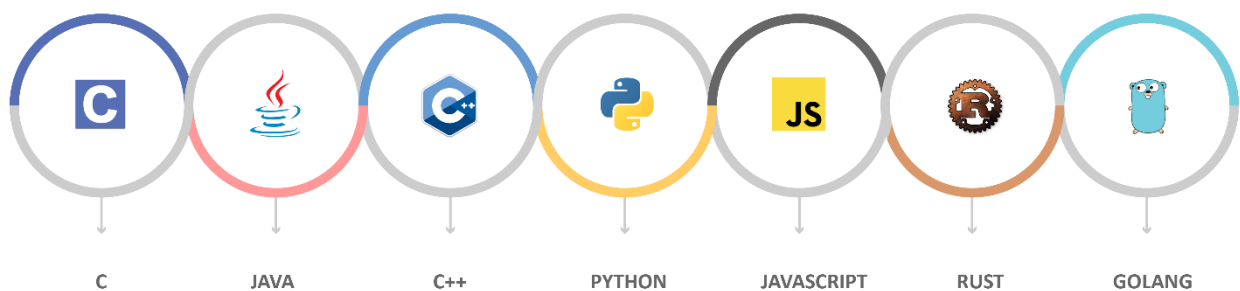
Computer software refers to the collection of programs, data, and instructions that enable a computer or computing device to perform various tasks and functions.

## What is Computer Programming?

Computer programming is the process of designing and writing computer programs.

## What are Programming Languages?

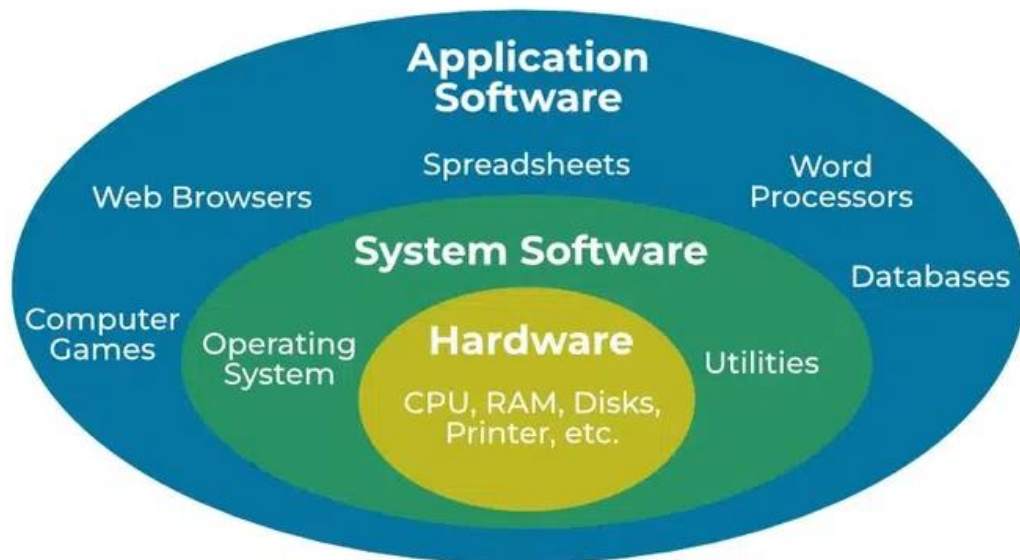
Computer language that is used by programmers (developers) to communicate with computers.



## Software Applications

1. System Software
2. Application Software
3. Networking and Web Applications Software
4. Embedded Software
5. Artificial Intelligence Software
6. Scientific Software

## Different between System and Application Software



| System Software  | Application Software   |
|--|--|
| System Software maintains the system resources and gives the path for application software to run.   | Application software is built for specific tasks.  |
| Low-level languages are used to write the system software.   | While high-level languages are used to write the application software.   |
| It is general-purpose software. Software which is designed to control, integrate and manage the individual hardware components and application software is known as system software. | It is a specific purpose software. Set of computer programs installed in the user's system and designed to perform a specific task is known as the application software. |
| System software runs independently.  | Application software is dependent on system software because they need platform for its functioning.   |
| System Software programming is more complex than application software.   | Application software programming is simpler in comparison to system software.  |
| Example: System software is an operating system, etc.  | Example: Application software is Photoshop, VLC player, etc.   |

## Basic Fundamental Concepts of Programming

- **Variable Declaration**

*Variables are names given to computer memory locations in order to store data in a program.*

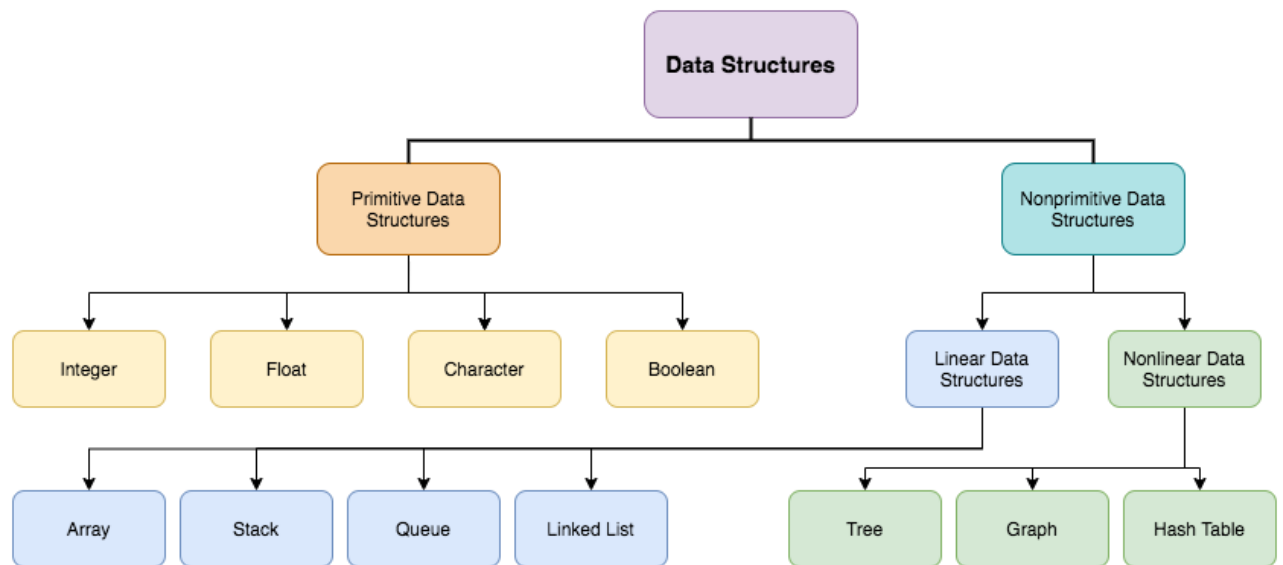
- **Basic Syntax**

*Syntax refers to the set of rules that define the structure of a language.*

- **Data Type and Structures**
- **Flow Control Structures (Conditionals and loops)**
- **Functional Programming**
- **Object-Oriented Programming Debugging**
- **IDEs and Coding Environments**

## Data Type and Structures

**Data Types** define the kind of information in programming (like numbers or words), while **Data Structures** organize and manage how that information is stored and used (like lists or tables).



## Flow Control Structures

Commands that allow a program to “decide” to take one direction or another.

Three basic types of control structures:

### 1. Sequential

*sequential control flow. It involves the execution of code statements one after the other.*

### 2. Selection

*selection flow control is, the computer decides what action to perform based on condition equaling true or false.*

### 3. Iteration

*A loop in programming repeatedly runs a statement or code block while a specified condition is true (evaluating to Boolean "true"), halting when the condition becomes false.*

## What is Functional Programming?

Functional programming is an approach to software development that uses pure functions to create maintainable software

## What are Pure Functions?

A function is called pure function if it always returns the same result for same argument

## What is Object Oriented Programming?

Object-Oriented Programming (OOP) is a programming concept that revolves around 'objects' and 'methods'

## What are the main concepts in Object Oriented Programming?

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

## What is the meaning of Debugging?

The process of identifying and removing errors from computer hardware or software.

## What is IDE?

IDE stands for integrated development environment, and it's a type of software that helps programmers create code. It does so by combining a number of functions into a single program, allowing users to write, test, and execute programs all from the same place, sometimes even with a graphical user interface.

## What is an Algorithm?

An algorithm is a step-by-step set of instructions or a precise process for solving a problem or completing a task, often used in computer programming and problem-solving.

## Characteristics of an algorithm

- Well-defined Inputs
- Well-defined Outputs
- Unambiguity

- Finiteness
- Effectiveness
- Language independence

What is Pseudocode?

A Pseudocode is defined as a step-by-step description of an algorithm.

Pseudocode Functionality

- **Variables**  
places to store values  
*quotient, decimalNumber, newBase*
- **Assignment**  
expression into a variable  
*Set quotient to 64*  
*quotient <-- 64*  
*quotient <-- 6 \* 10 + 4*
- **Output**  
Printing a value  
*Write, Print*
- **Input**  
Getting values  
*Get, Read*
- **Selection**  
choice to execute or skip a statement  
*Read number*  
*If (number < 0)*  
*Write number + " is less than zero."*  
*or*  
*Write "Enter a positive number."*

## Two methodologies of Developing an Algorithm

- Top-down design
- Object-oriented design

### Top-Down Design

Problem-solving technique in which the problem is divided into sub problems; the process is applied to each sub problem

### Object-oriented Design


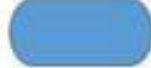






A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called objects

### What are The Flowcharts?

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.



## Flowchart Symbols

| Symbol  | Symbol Name             | Description  |
|---|-------------------------|--|
|    | Flow lines              | Flow lines are used to connect symbols used in flowchart and indicate direction of flow.                               |
|    | Terminal (START / STOP) | This is used to represent start and end of the flowchart.  |
|    | Input / Output          | It represents information which the system reads as input or sends as output.  |
|    | Processing              | Any process is represented by this symbol. For example, arithmetic operation, data movement.                           |
|    | Decision                | This symbol is used to check any condition or take decision for which there are two answers. Yes (True) or No (False). |
|   | Connector               | It is used to connect or join flow lines.  |
|  | Off-page Connector      | This symbol indicates the continuation of flowchart on the next page.  |
|  | Document                | It represents a paper document produced during the flowchart process.  |

## What are the 03 Development Techniques?

### - Modular Programming

*Modular programming is an approach in software development where a program is divided into smaller, **self-contained modules** or pieces, **each responsible for a specific function**, making the code more organized, maintainable, and easier to understand.*

### - Defensive Programming

*Defensive programming is a coding approach that focuses on **anticipating and handling potential errors, exceptions, and unexpected situations** in software development to ensure more reliable and robust programs.*

- **Recursion**

*Recursion is a programming technique where a function calls itself to solve a problem by **breaking it down into smaller instances of the same problem**, often used in tasks that have a **repeating structure** or **can be expressed** in terms of simpler versions of themselves.*

## Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a **structured process** that **guides the creation and management** of software applications, involving stages from planning and design to testing and maintenance. It ensures a systematic approach to development.

### Stages in Software Development Life Cycle

- **Planning and Requirement Analysis**

*Gathering project requirements and planning the software development process.*

- **Defining Requirements**

*Defining detailed specifications and features based on gathered requirements.*

- **Designing the Software**

*Creating a blueprint of the software's architecture, user interfaces, and components.*

- **Developing the Project**

*Writing and coding the actual software according to the design.*

- **Testing**

*Rigorously testing the software to identify and fix errors, bugs, and issues.*

- **Deployment**

*Releasing the software for users to access and use.*

- **Maintenance**

*Continuously monitoring, updating, and enhancing the software to ensure its functionality and performance over time.*

## SDLC Models

- **Waterfall Model**

*A linear approach where each phase of the software development process follows the previous one sequentially.*

- **RAD Model (Rapid Application Development)**

*Focuses on rapid prototyping and quick iterations to expedite development.*

- **Prototype Model**

*Involves creating an initial prototype to better understand user requirements before final development.*

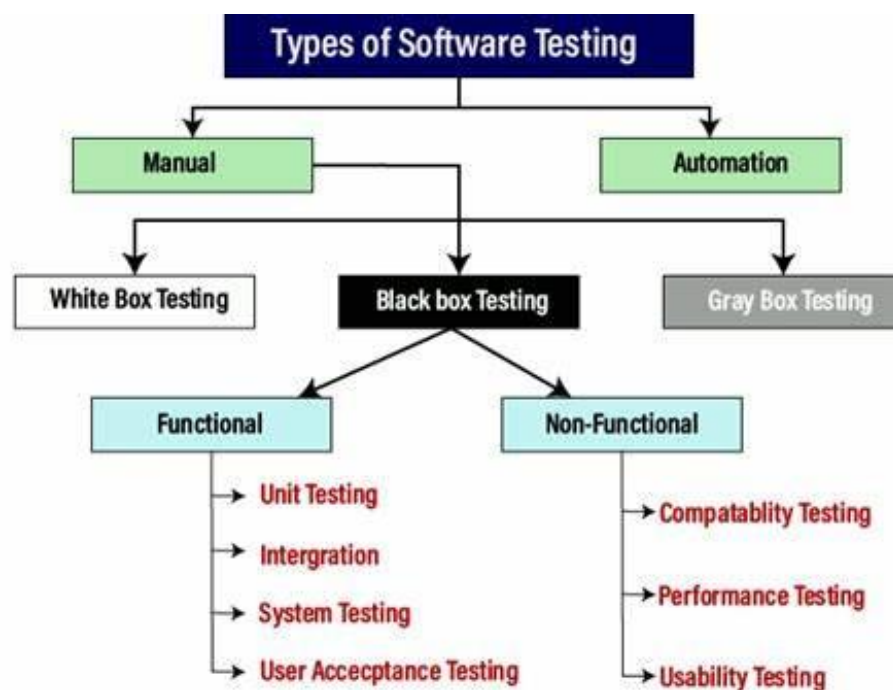
- **Agile Model**

*Emphasizes flexibility and collaboration, with iterative cycles of development and continuous user feedback.*

## What is Software Testing?

**Software testing is checking a program to find and fix mistakes before it's used by people.**

## Types of Software Testing



- **Manual Testing**

*Human testers manually run the software, performing tests and checks **without automated tools**.*

*Three Types of Manual Testing*

- **White box testing**

Involves examining the internal logic and code structure of the software.

- **Black box testing**

Focuses on testing the software's functionality without knowledge of its internal code.

- **Gray box testing**

Combines elements of both **white box and black box** testing, where some internal details are known but not all.

- **Automation Testing**

*Testing is done using **automated tools and scripts** to increase efficiency and repeatability of tests.*

Automation Testing Tools

## The 10 Best Test Automation Tools



endtest



Playwright



cucumber

## What is Dry Run Testing?

**Dry Run Testing** is a manual method where testers simulate code execution to find logic errors without actually running the program.

## What is the meaning of Test Case?

Test cases are detailed instructions that describe how to test a software application to ensure it works properly.

## What is Test Scenario?

A test scenario is a collection of related test cases that together verify a specific aspect of a software application.

## Example Test Case

| Test Case ID | Description          | Test Steps                           | Expected Results                | Actual Results | Status     |
|--------------|----------------------|--------------------------------------|---------------------------------|----------------|------------|
| TC001        | Login Functionality  | 1. Enter valid username and password | User is logged in successfully  |                | Incomplete |
|              |                      | 2. Click on 'Login' button           |                                 |                |            |
| TC002        | Registration Process | 1. Fill in valid user details        | User is registered successfully |                | Incomplete |
|              |                      | 2. Submit registration form          |                                 |                |            |

## What is Documentation in Software Development?

Documentation in software development involves creating written materials to explain the software's design, usage, and maintenance, aiding developers and users in understanding and interacting with the software effectively.

## General requirements of all software documentation

- Facilitating communication within the team.
- Serving as an information source for maintenance engineers.
- Providing management with details for program management.
- Describing system operation and administration to users.
- Creating documentation before coding (e.g., design docs).
- Continuing documentation post-code (e.g., user manuals).

## The two main types of documentation

- Process Documents

*These detail the **steps, methodologies, and guidelines** used **during** software development.*

*Examples: Project schedules, coding standards, testing procedures*

- **Product documents**

*These describe the software's **features, functionalities, and usage instructions** for **end-users and stakeholders**.*

*Examples: User manuals, technical specifications, API documentation.*

## What is Document Quality?

Document quality encompasses accuracy, comprehensiveness, and clarity, ensuring effective communication and usability.

## What is Document Structure?

Document structure outlines how information is organized within software documentation, ensuring logical flow and easy navigation for readers.

## Best Practices for Document Structure

- Include a cover page for all documents.
- Organize documents into chapters, sections, and subsections.
- Incorporate an index for extensive reference material.
- Integrate a glossary to clarify ambiguous terms.

## What does the IEEE Standard for User Documentation recommend regarding the structure of documentation?

The IEEE Standard for User Documentation advises a structured approach including a cover page, divided chapters, an index, and a glossary.

## Online Documentation

Online documentation offers digital resources accessible via the internet, providing information and support for various topics, including software applications.

## Document Storage

- **Author's Recommendation (in 2001):**

*Use a File System to store actual documents.*

*Utilize a Database to store file references along with metadata for search and reference capabilities.*

- **Modern Approach (Today):**

*Employ Content Management Systems (CMS) like CVS or Subversion.*

*These are free, open-source, and simple to set up and manage.*

## What is Software Quality?

Software quality is the measure of how well a software application meets requirements and user expectations, considering Software Quality Attributes.

### Software Quality Attributes

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability

## Software Quality Assurance

Software Quality Assurance is a structured approach to consistently ensure the development and maintenance of

high-quality software by following best practices and processes.

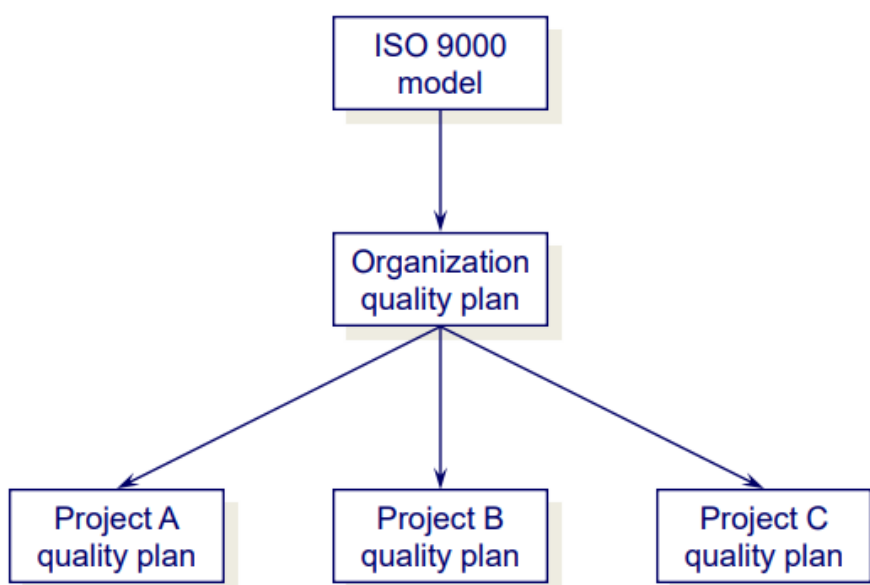
### Three-Prong Approach to Quality Management

- Establishing Organization -wide Policies  
*Organizations set overarching policies, procedures, and standards to uphold quality consistently.*
- Tailoring Project-specific Policies  
*From the established templates, organizations customize project-specific policies and procedures.*
- Quality Control  
*Vigilant monitoring is crucial, ensuring adherence to appropriate procedures for each project.*

### Standards for Quality Assurance

- **ISO 9000-3:** *This standard guides organizations in creating effective software quality assurance plans.*
- **ANSI/IEEE Standards:** *These standards provide guidelines for software quality processes.*

### A Software Quality Plan





## Steps to Develop and Implement Software Quality Assurance Plan

This slide is 100% editable. Adapt it to your needs and capture your audience's attention.



### SQA Activities

- Applying technical methods for high-quality specifications and design.
- Formal technical reviews for uncovering quality problems.
- Effective software testing through various design methods.
- Enforcing standards and controlling change.
- Measurement to track quality and assess procedural changes.
- Record keeping and reporting to ensure proper information dissemination.

### Advantages of Software Quality Assurance (SQA)

- **Higher Quality**  
*SQA ensures consistent, high-quality software products.*
- **Reduced Defects**  
*Early issue identification leads to fewer defects.*
- **Cost Savings**  
*Less rework and customer support costs.*
- **Higher Customer Satisfaction**  
*Improved product reliability boosts satisfaction.*
- **Efficient Development**

*Standardized processes lead to efficiency.*

- **Risk Mitigation**

*SQA minimizes project risks and uncertainties.*

- **Compliance**

*Ensures adherence to industry standards and regulations.*

## Disadvantages of Software Quality Assurance (SQA)

- **Increased Overhead**

*SQA processes require time and resources.*

- **Complex Implementation**

*Setting up SQA practices can be complex.*

- **Resistance to Change**

*Teams might resist new procedures.*

- **Costly**

*Initial setup and training costs can be significant.*

- **Overemphasis on Process**

*Focusing solely on process might stifle creativity.*

- **Potential Delays**

*Stringent processes might slow down development.*

- **False Sense of Security**

*Relying solely on SQA doesn't guarantee defect-free software.*

## What Is Software Security?

**Software and security concern the protection of software applications and systems against unauthorized access, data breaches, vulnerabilities, and cyber threats, ensuring the confidentiality, integrity, and availability of information while maintaining user privacy and trust.**

## Starting Point for Ensuring Security

**Before addressing security, it's essential to conduct an inventory of key factors**

- **Stakeholders:** Identify owners, companies, or parties involved.
- **Assets:** List data, services, customer information, etc., that need protection.
- **Threats:** Understand potential risks like data breaches, theft, etc.
- **Attackers:** Recognize potential sources, from employees to criminals.

## Security Concepts

- Security Policy

A security policy outlines the desired security requirements and goals of implemented countermeasures. It answers questions like, "Secure against what and from whom?"

- Enforcing Security

What are the three fundamental security objectives

- Confidentiality: Ensuring that information is accessible only to authorized individuals and remains private from unauthorized access.
- Integrity: Safeguarding data from unauthorized alteration or tampering, maintaining its accuracy and reliability.
- Availability: Ensuring that information and resources are accessible and usable when needed, minimizing downtime or disruption.

## Security Goals

While the CIA trio (confidentiality, integrity, availability) is widely recognized, additional security goals include:

- Traceability and Auditing: Enabling forensic analysis and tracking actions for accountability.
- Monitoring: Real-time auditing to swiftly identify anomalies and potential threats.
- Multi-Level Security: Implementing varying access levels based on user roles.
- Privacy & Anonymity: Safeguarding personal information and ensuring user anonymity.
- Assurance: Ensuring that security goals are met, often termed as "information assurance."

## What is Cloud Computing?

Cloud computing involves accessing and using computing resources and services over the internet, providing scalability and flexibility for various tasks.

## Types of Cloud Computing

- **Private clouds**  
Resources are dedicated to a single organization, providing enhanced control and privacy.
- **Public clouds**  
Services are available to the general public, managed by third-party providers, offering scalability and cost-effectiveness.
- **Hybrid clouds**  
Combining private and public clouds to balance flexibility and security.
- **Multi clouds**  
Using multiple cloud services from different providers to avoid vendor lock-in and optimize features.

## Cloud Computing Services

- Infrastructure as a Service (IaaS): Provides virtualized computing resources like virtual machines, storage, and networking.
- Platform as a Service (PaaS): Offers a platform with tools and services for application development, testing, and deployment.
- Software as a Service (SaaS): Delivers software applications over the internet, eliminating the need for installation and allowing access from various devices.

## Program Structure: A Common Pattern

- **Start Statements:** Initial statements that mark the beginning of the program.
- **Variable Declaration:** Defining variables to hold data.
- **Program Statements:** Organized blocks of code that perform specific tasks.

## Variable Declaration

Variable declaration in programming involves specifying the **type and name of a variable that will store data during program execution. It allocates memory space for the variable to hold values of a particular data type, such as numbers, text, or objects.**

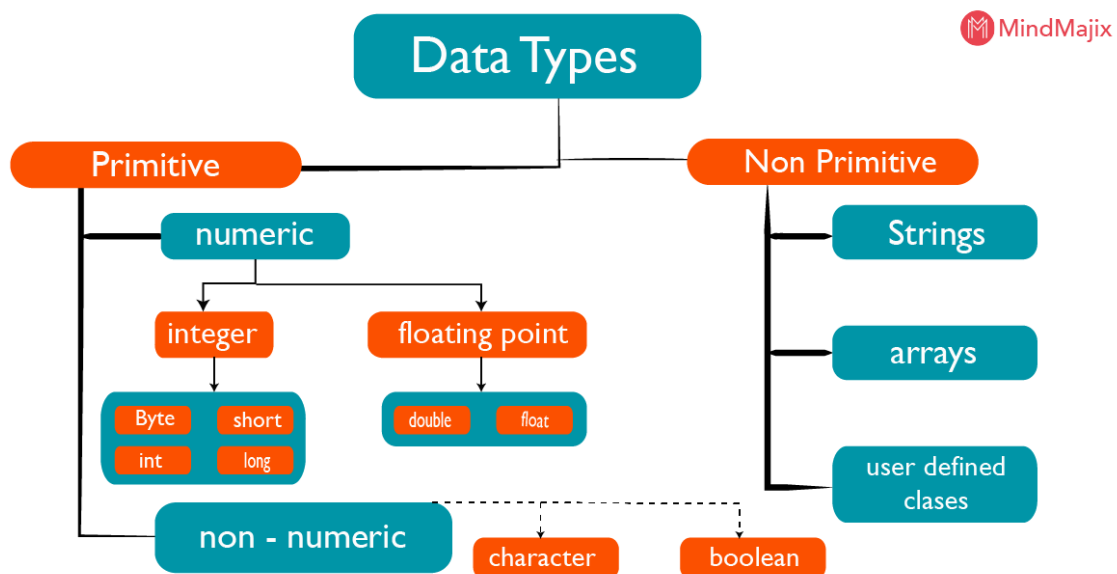
## What is the meaning of Data Types?

Data types in programming define the kind of data that a variable can hold. Common data

## Common Data Types

- Integer: Whole numbers.
- Float: Decimal numbers.
- String: Text.
- Boolean: True or false values.
- Array: Collection of elements.
- Object: Complex data structure.

## Classification of Data Types



## Primitive Data Types and Non-Primitive Data Types

- **Primitive Data Types**

*Basic built-in types (Integer, Float, Character, Boolean) used for storing simple values **directly**.*

- **Non-Primitive Data Types**

*Complex types (Arrays, Objects, Strings) created using **primitive or other non-primitive types**, referring to memory locations rather than holding the value directly.*

There are eight primitive data types

| Data Type | Size    | Description   |
|-----------|---------|---|
| byte      | 1 byte  | Stores whole numbers from -128 to 127   |
| short     | 2 bytes | Stores whole numbers from -32,768 to 32,767                                       |
| int       | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647                         |
| long      | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float     | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits           |
| double    | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits               |
| boolean   | 1 bit   | Stores true or false values   |
| char      | 2 bytes | Stores a single character/letter or ASCII values                                  |

Numeric Data Types: **Numbers used for mathematical computations, like integers and decimals.**

Non-Numeric Data Types: **Data that can't be mathematically manipulated, including text (strings), dates, and boolean values (true/false)**

Elementary/Fundamental Data Types

**Basic building blocks in programming**, like integers, floating-point numbers, characters, and booleans, defining the kind of data variables can hold.

## Derived Data Types

Derived types, like arrays, structures, and classes, are created by **combining primitive or other derived types**, enabling the creation of custom data structures.

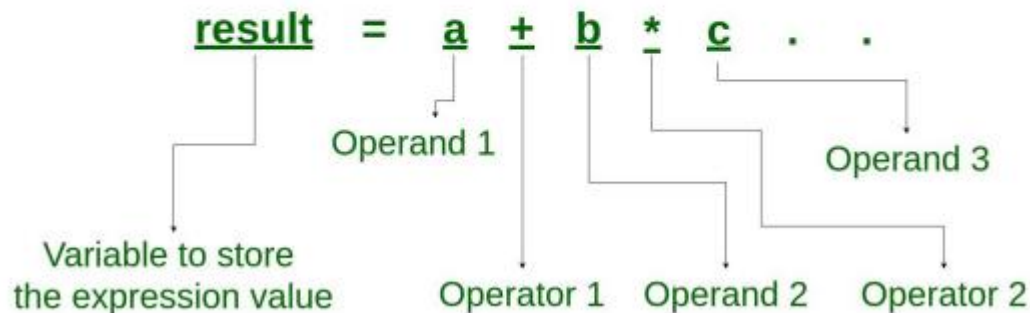
## Different between Derived and Elementary Data Types

| Aspect                   | Fundamental Data Types  | Derived Data Types                  |
|--------------------------|-------------------------|-------------------------------------|
| <b>Definition</b>        | Basic building blocks   | Created by combining types          |
| <b>Examples</b>          | Integers, floats, chars | Arrays, structures, classes         |
| <b>Support</b>           | Directly supported      | Constructed using fundamentals      |
| <b>Complexity</b>        | Simple                  | More complex for customization      |
| <b>Usage</b>             | Simple data             | Custom data structures              |
| <b>Memory Allocation</b> | Direct allocation       | May involve more complex allocation |

## What is an Expression?

Expressions are combinations of values, operators, and variables that yield a result when evaluated in programming.

# What is an Expression?



## Types of Expressions

- **Constant expressions**  
Consist of constant values and operators.  
Examples: 5, 10 + 5 / 6.0, 'x'
- **Integral expressions**  
*Involve integer values and operators.*  
Examples: `x`, `x * y`, `x + int(5.0)`
- **Floating expressions**  
*Deal with floating-point values and operators.*  
Examples: `x + y`, 10.75
- **Relational expressions**  
*Compare values using relational operators.*  
Examples: `x <= y`, `x + y > 2`
- **Logical expressions**  
*Combine conditions using logical operators.*  
Examples: `x > y && x == 10`, `x == 10 || y == 5`
- **Pointer expressions**  
*Involve pointers and pointer arithmetic.*  
Examples: `&x`, `ptr`, `ptr++`

### Bitwise expressions

*Perform operations at the bit level.*



*x << 3 shifts three-bit position to left*

## What Is a Subprogram?

A subprogram, like a function or subroutine, is a distinct and reusable block of code that performs a specific task within a larger program.

## Two varieties in Subprogram

### - Functions

*Subprograms that return a value after performing a specific task.*

Example:

```
def square(number):  
    return number * number
```

```
result = square(5)  
print("Square:", result)
```

### - Procedures

*Subprograms that execute a sequence of statements without returning a value, often used for their side effects.*

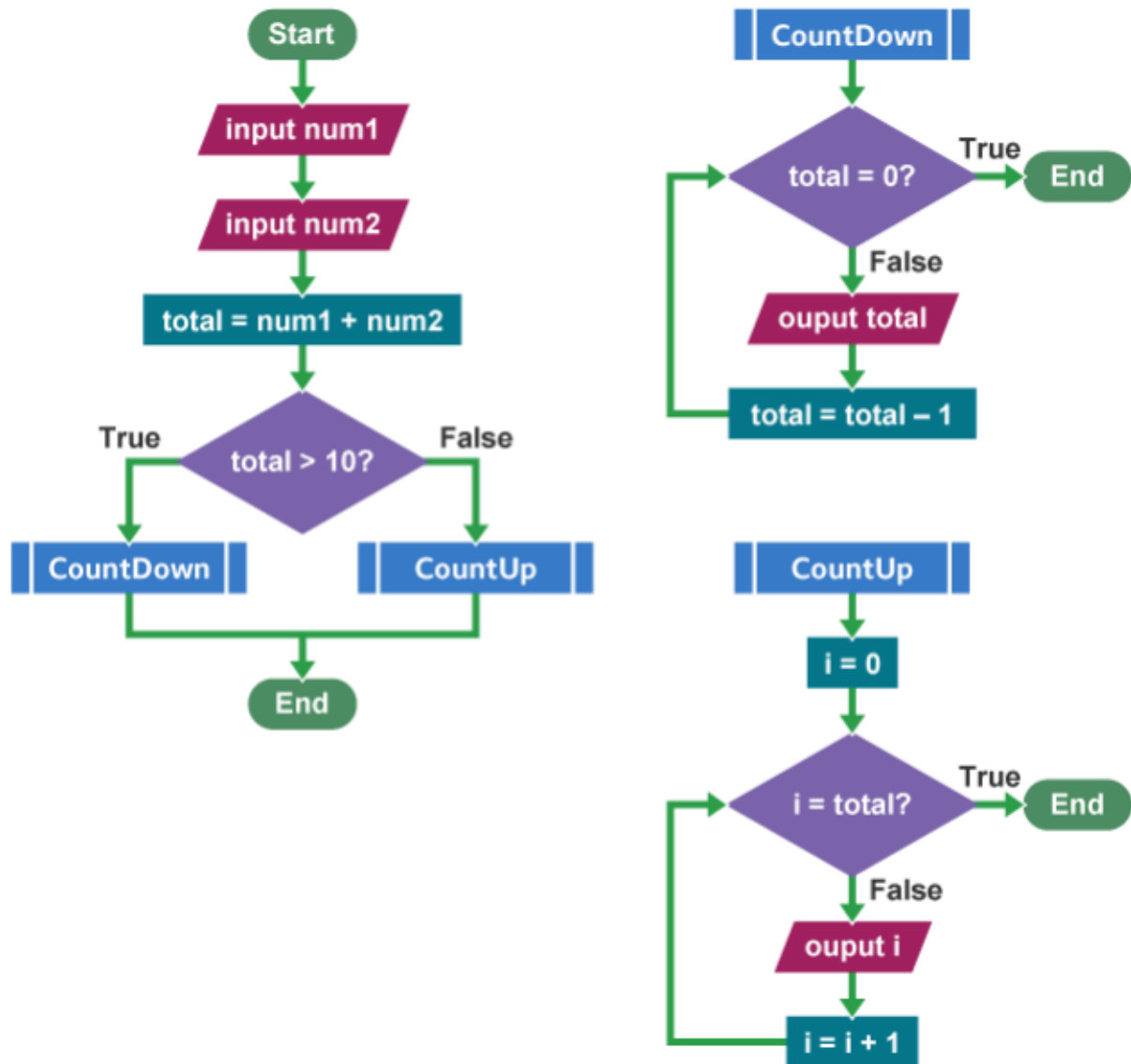
Example:

```
def greet(name):  
    print("Hello,", name, "! Welcome!")
```

```
name = "Alice"  
greet(name)
```

## What are the Subroutines?

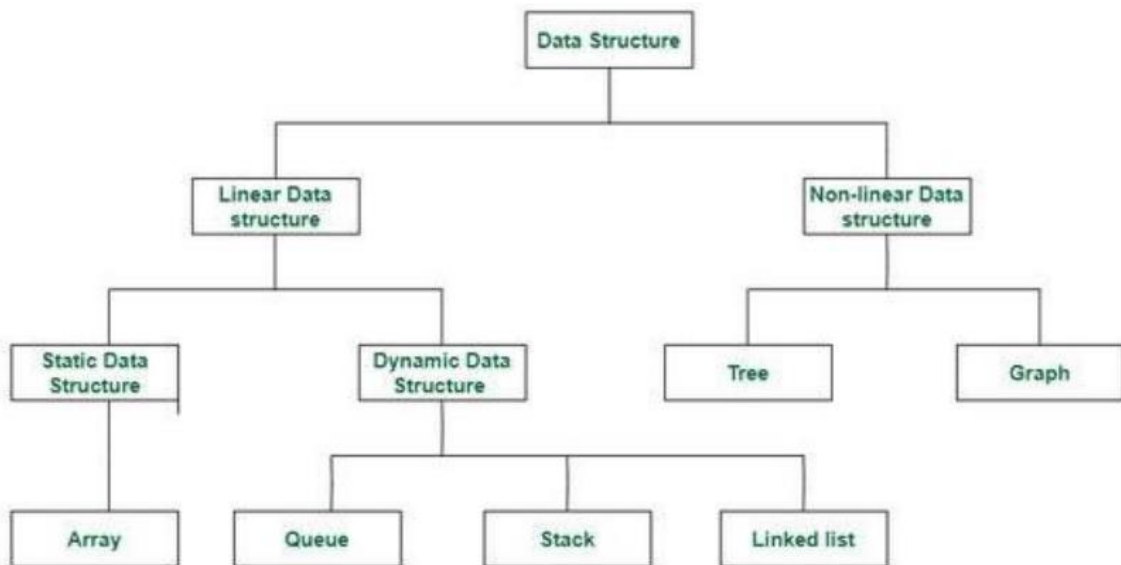
Subroutines are specific code blocks in a program that accomplish tasks, including functions (returning values) and procedures (executing tasks).



## What is a Data Structure?

A data structure is a method of organizing and storing data effectively to enable easier manipulation and access in programming.

## Classifications of Data Structure



### Linear Data Structure

A linear data structure organizes data elements sequentially, facilitating sequential access.

#### Types of Linear Data Structures

- **Static data structure**  
*Has **a fixed size** determined during declaration, like static arrays.*
- **Dynamic data structure**  
*Can **change in size during runtime**, such as linked lists or dynamic arrays, accommodating varying amounts of data.*

### Non-linear Data Structure

Non-linear data structures, like **trees and graphs**, arrange data in **complex interconnected patterns**, unlike linear structures such as arrays or linked lists.

### Major Operations in Data Structures

- **Searching**: Locating a specific element within the data structure.
- **Sorting**: Arranging elements in a specific order, often ascending or descending.

- **Insertion:** Adding new elements to the data structure.
- **Updating:** Modifying existing elements with new values.
- **Deletion:** Removing elements from the data structure.

## Advantages of Data Structures

- **Efficiency:** Well-chosen data structures optimize operations like searching, insertion, and sorting, improving program performance.
- **Reusability:** Defined data structures can be reused across different parts of a program or in multiple projects.
- **Abstraction:** Data structures abstract complex data arrangements, making programming tasks more manageable and intuitive.

## Abstract Data Type

**Abstract Data type (ADT)** is a type (or class) for objects whose behavior is defined by a set of values and a set of operations.

## Collections

**Collections** are abstract data types that define how data is structured and accessed, without dictating a specific implementation.

**Collections** often establish specific linear orders, granting access to one or both ends. The underlying data structure might not be linear; for instance, a priority queue can be implemented using a heap, which is a tree structure.

## Significant Linear Collections

- **Lists:** Sequential arrangement of elements.
- **Stacks:** Follows Last-In-First-Out (LIFO) order.
- **Queues:** Follows First-In-First-Out (FIFO) order.
- **Priority Queues:** Elements have assigned priorities.
- **Double-Ended Queues:** Allows insertion and removal at both ends.
- **Double-Ended Priority Queues:** Combines priorities and double-ended properties.

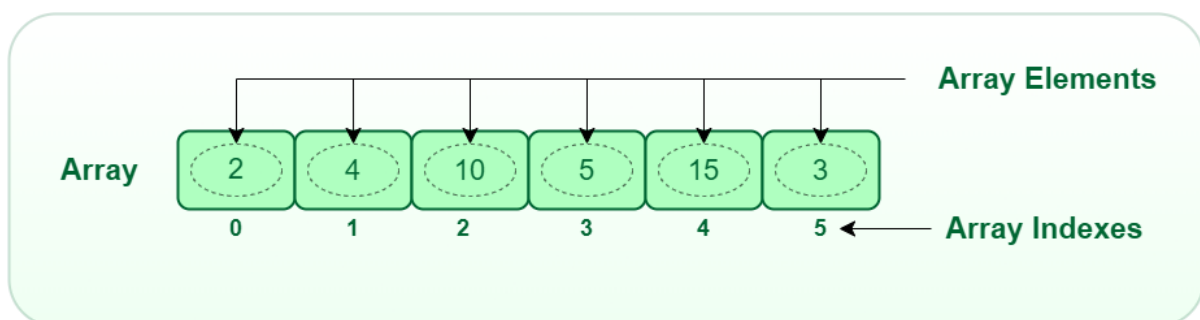
# Arrays

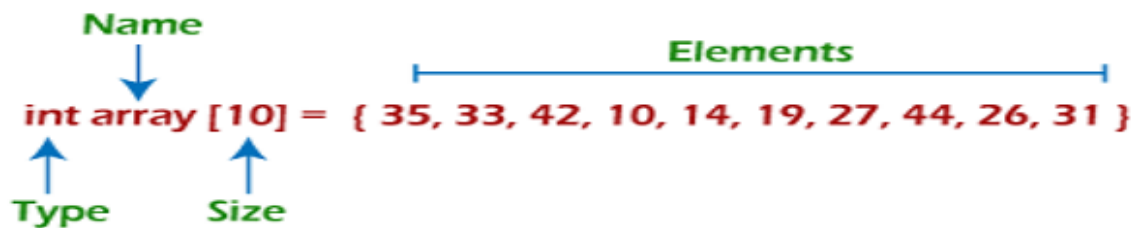
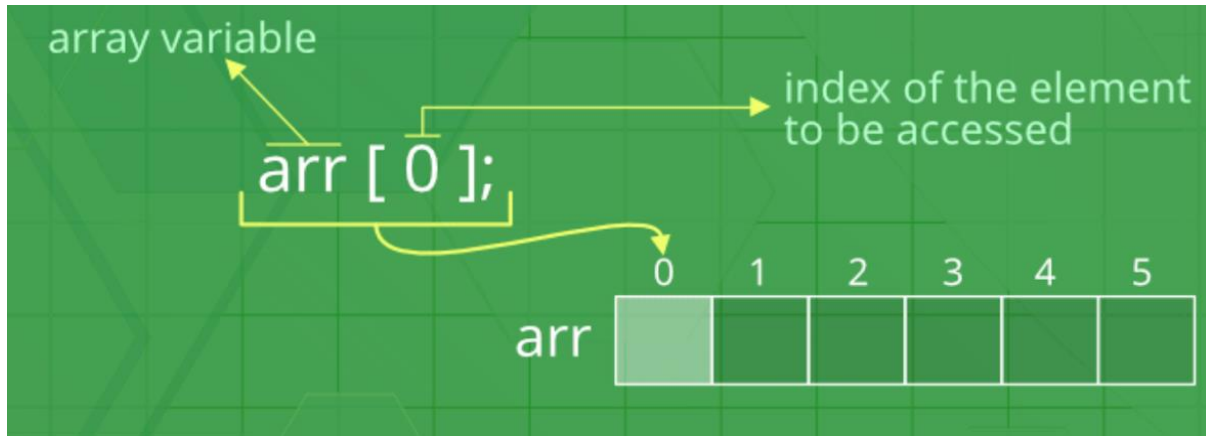
Arrays are linear data structures storing elements in sequential memory locations, accessed via indices, useful for data organization, but their fixed size can be limiting in dynamic scenarios.

## Properties of Array

- **Fixed Size:** Arrays have a predetermined size set during creation that doesn't change during runtime.
- **Contiguous Memory:** Elements are stored in adjacent memory locations, enabling efficient access.
- **Indexed Access:** Elements are accessed using indices, starting from 0 for the first element.
- **Homogeneous Elements:** Arrays store elements of the same data type.
- **Direct Addressing:** Elements can be directly accessed using their index
- **Static Data Structure:** The size is known at compile-time, making them static data structures.

## Representation of an Array





## Basic Operations in the Arrays

### - Insertion

*Adding an element at a specific position in the array.*

#### Algorithm

1. Start
2. Create an Array of a desired datatype and size.
3. Initialize a variable 'i' as 0.
4. Enter the element at ith index of the array.
5. Increment i by 1.
6. Repeat Steps 4 & 5 until the end of the array.
7. Stop

### - Deletion

*Removing an element from a specific position in the array.*

#### Algorithm

1. Start
2. Set  $J = K$
3. Repeat steps 4 and 5 while  $J < N$
4. Set  $LA[J] = LA[J + 1]$
5. Set  $J = J + 1$
6. Set  $N = N - 1$
7. Stop

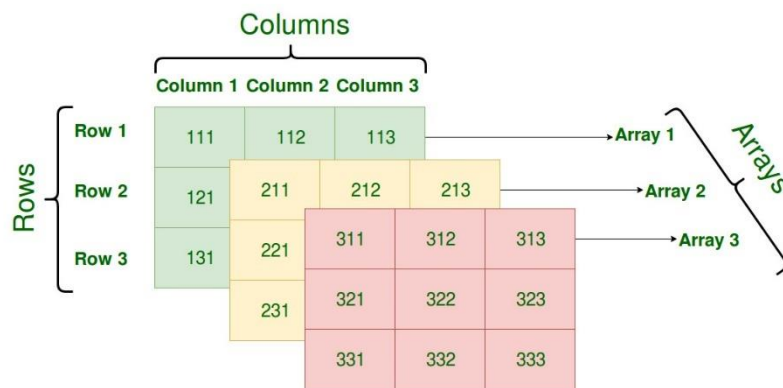
- **Search**  
*Removing an element from a specific position in the array.*
- **Update**  
*Modifying the value of an element at a specific position in the array.*
- **Traverse**  
*Visiting each element in the array sequentially.*
- **Display**  
*Printing the elements of the array for viewing.*

## 2D Arrays

A grid of elements organized in rows and columns for handling structured data that needs two dimensions for arrangement and access.

## 2D Array Representation

```
int arr[max_rows][max_columns];
```

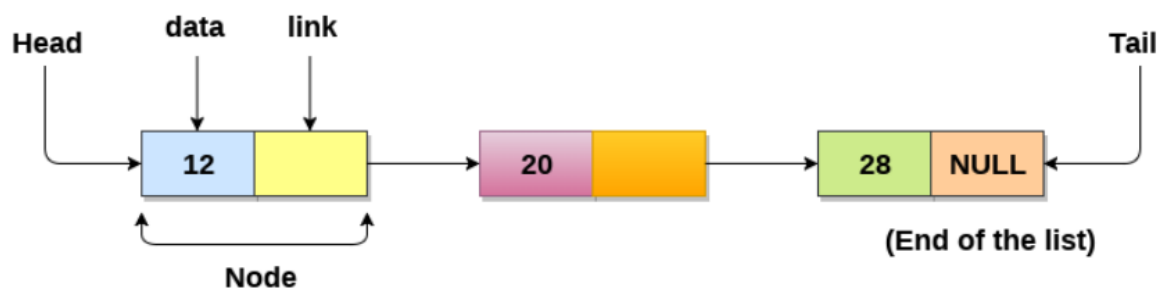


## Advantages and Disadvantages of Arrays

| Advantages of Arrays  | Disadvantages of Arrays |
|-----------------------|-------------------------|
| Fast Access           | Fixed Size              |
| Sequential Storage    | Memory Allocation       |
| Simple Implementation | Insertion/Deletion      |
|                       | Sparse Data Handling    |
|                       | Static Nature           |

## Linked Lists

A linear data structure composed of nodes, each containing data and a pointer to the next node, allowing dynamic memory allocation, efficient insertion, and deletion operations.



## Properties of Linked List

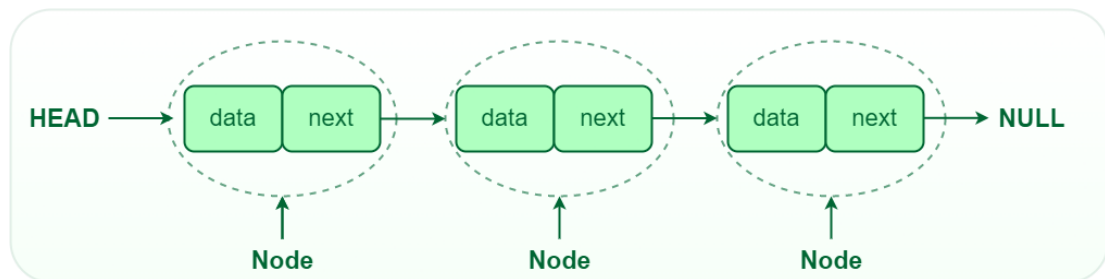
- **Head:** The starting node of the linked list.
- **Tail:** The last node of the linked list.
- **Data:** The value stored in each node.
- **Node:** A unit containing data and a reference to the next node.
- **Link/Pointer:** The reference connecting nodes.



## Types Of Linked List

- **Single-linked list**

Each node points to the next node, forming a unidirectional sequence.



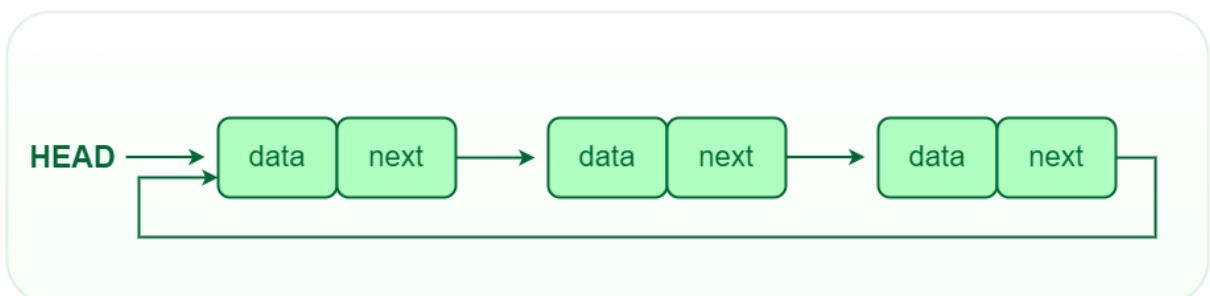
- **Double linked list**

Each node points to both the next and the previous node, allowing bidirectional traversal.



- **Circular linked list**

The last node points back to the first node, forming a circular structure. It can be single or double linked.



## Uses of Linked Lists

- *Dynamic Memory Allocation:* Adaptable memory usage for changing data sizes.
- *Insertion and Deletion:* Efficient operations for adding or removing elements.
- *Data Structure Implementation:* Building blocks for advanced structures like stacks and queues.
- *Memory Efficiency:* Reduced wastage due to non-contiguous memory allocation.
- *Sparse Data Handling:* Effective for data with many gaps or empty spaces.
- *Infinite Data Streams:* Suitable for representing continuous or large data streams.
- *In-place Reversal:* Efficient reversal of elements within the list.

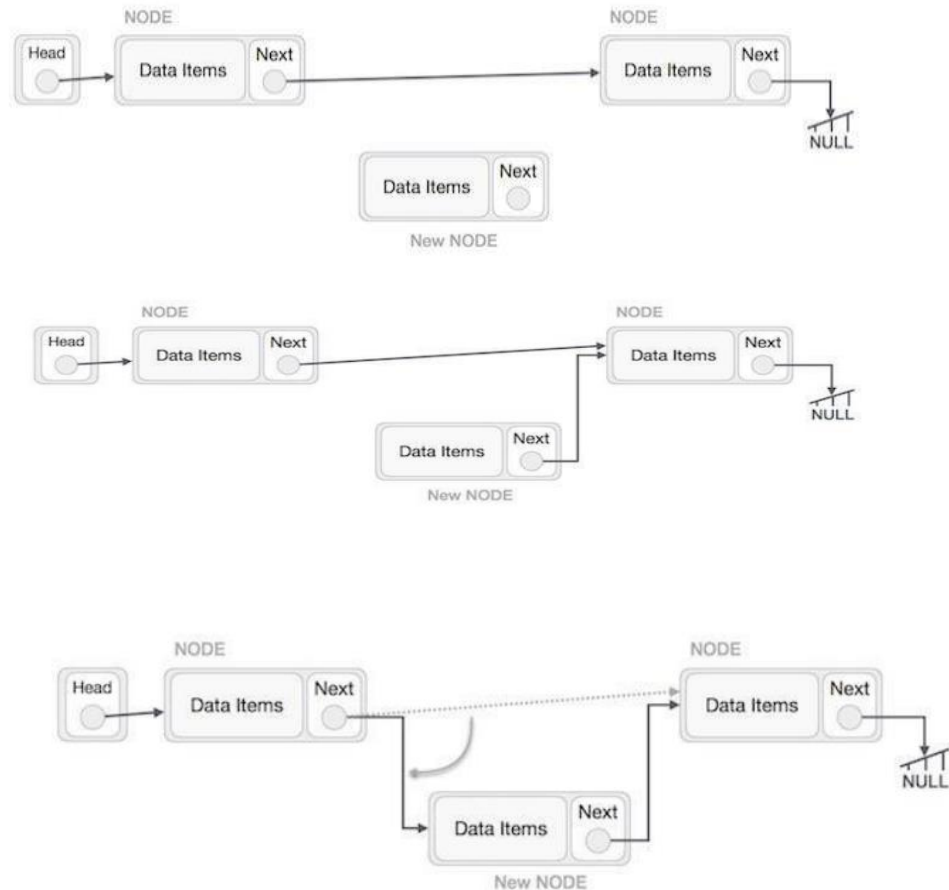
## Use Linked List Over Array When

- *Dynamic Size:* Linked lists accommodate changing data sizes, while arrays have fixed sizes.
- *Insertion and Deletion:* Linked lists efficiently handle element insertion and deletion, avoiding costly shifts in arrays.
- *Memory Efficiency:* Linked lists allocate memory dynamically, reducing wastage compared to fixed-sized arrays.
- *Sparse Data:* Linked lists suit scenarios with many gaps or empty spaces in data.
- *Infinite Data:* Linked lists represent continuous or large data streams more effectively.
- *Efficient Reversal:* Linked lists can be reversed in-place without consuming extra memory.

## Basic Operations in the Linked Lists

### - Insertion

*Adding a new node to the linked list at a specific position.*



### Three Types Insertion

#### Algorithm for Insertion at Beginning

1. **START**
2. *Create a node to store the data*
3. *Check if the list is empty*
4. *If the list is empty, add the data to the node and assign the head pointer to it.*
5. *If the list is not empty, add the data to a node and link to the current head. Assign the head to the newly added node.*
6. **END**

#### Algorithm for Insertion at Ending

1. *START*
2. *Create a new node and assign the data*
3. *Find the last node*
4. *Point the last node to new node*
5. *END*

#### Algorithm for Insertion at a Given Position

1. *START*
2. *Create a new node and assign data to it*
3. *Iterate until the node at position is found*
4. *Point first to new first node*
6. *END*

### - Deletion

*Removing a node from the linked list at a specific position.*

#### Three Types Deletion

##### Algorithm for Deletion at Beginning

1. *START*
2. *Assign the head pointer to the next node in*
3. *the list*
4. *END*

##### Algorithm for Deletion at Ending

1. *START*
2. *Create a new node and assign the data*
3. *Find the last node*
4. *Point the last node to new node*
5. *END*

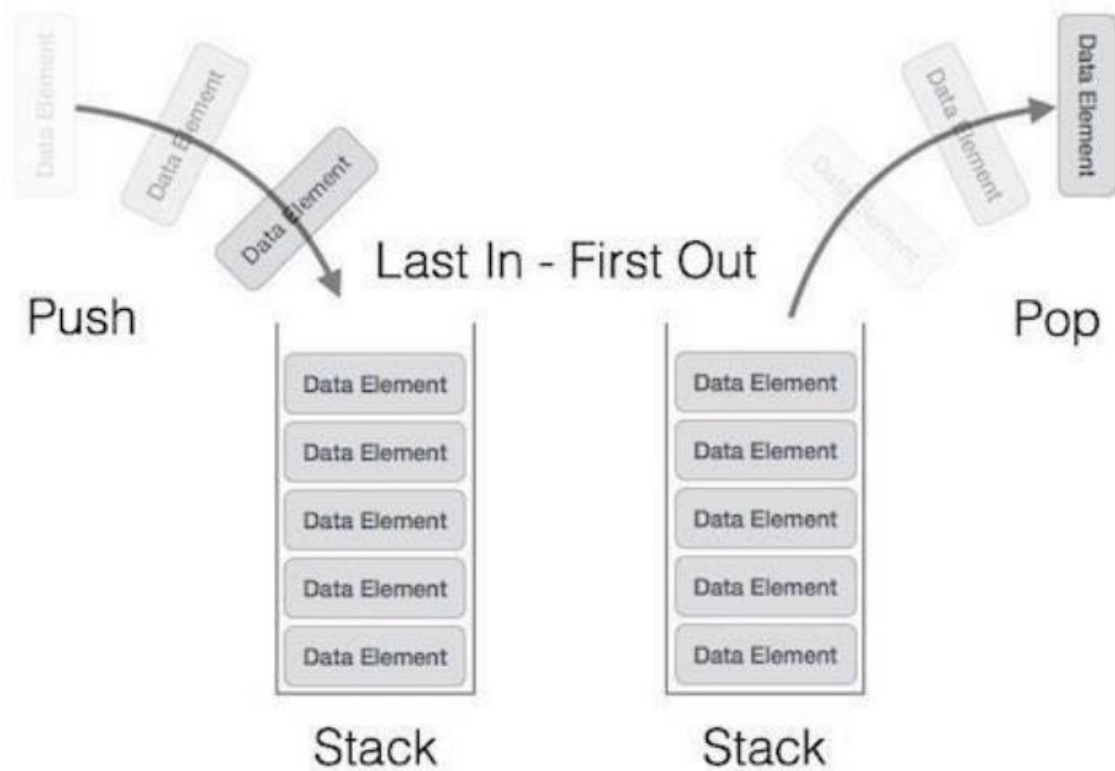
##### Algorithm for Deletion at a Given Position

1. *START*
2. *Iterate until find the current node at position in the list*
3. *Assign the adjacent node of current node in the list to its previous node.*
4. *END*

- Display
- Search
- Delete

## Stack

A stack is a data structure based on the Last-In-First-Out (LIFO) principle, used for managing elements where the last added is the first processed, commonly used in function calls and expression evaluation.



## Basic Operations on Stacks

- **push():** Adds an element to the top of the stack.
- **pop():** Removes and returns the top element from the stack.
- **peek():** Returns the top element without removing it.
- **isFull():** Checks if the stack is full and can't accommodate more elements.
- **isEmpty():** Checks if the stack is empty and contains no elements.

### Insertion: push()

#### Algorithm

1. - Checks if the stack is full.
2. - If the stack is full, produces an error and exit.
3. - If the stack is not full, increments top to point next empty space.
4. - Adds data element to the stack location, where top is pointing.
5. - Returns success.

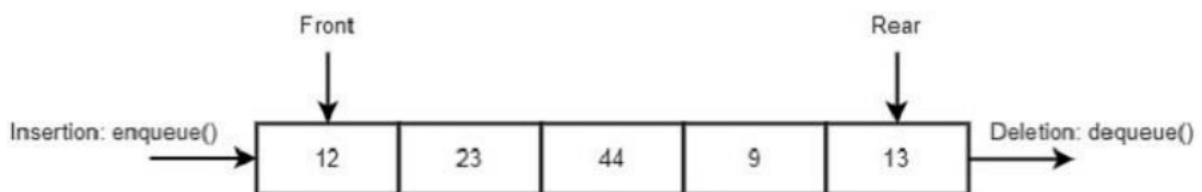
### Deletion: pop()

#### Algorithm

1. Checks if the stack is empty.
2. If the stack is empty, produces an error and exit.
3. If the stack is not empty, accesses the data element at which top is pointing.
4. Decreases the value of top by 1.
5. Returns success.

## Queue

A queue is a linear data structure following the First-In-First-Out (FIFO) rule, useful for managing elements in the order they arrive, often used in task scheduling and job processing.



Queue: FIFO Operation

## Basic Operations

- *enqueue(): Adds an element to the back of the queue.*
- *dequeue(): Removes and returns the front element from the queue.*

### Insertion Operation: enqueue()

#### Algorithm

1. *START*
2. *Check if the queue is full.*
3. *If the queue is full, produce overflow error and exit.*
4. *If the queue is not full, increment rear pointer to point the next empty space.*
5. *Add data element to the queue location, where the rear is pointing.*
6. *return success.*
7. *END*

### Deletion Operation: dequeue()

#### Algorithm

1. *START*
2. *Check if the queue is empty.*
3. *If the queue is empty, produce underflow error and exit.*
4. *If the queue is not empty, access the data where front is pointing.*
5. *Increment front pointer to point to the next available data element.*
6. *Return success.*
7. *END*

## What is Data Abstraction?

Data abstraction simplifies usage by hiding complex details, focusing on essential features, enhancing software modularity, and improving code manageability and understandability.

## Types of Algorithms

- **Sorting Algorithms**  
Algorithms that arrange elements in a particular order, such as ascending or descending,

## Types of Sorting Algorithms

### – Selection Sort

Repeatedly selects the minimum element from the unsorted portion and places it at the beginning.

*Algorithm:*

*Step 1 – Set MIN to location 0*

*Step 2 – Search the minimum element in the list*

*Step 3 – Swap with value at location MIN*

*Step 4 – Increment MIN to point to next element*

*Step 5 – Repeat until list is sorted*

### – Bubble Sort

Repeatedly compares adjacent elements and swaps if they're in the wrong order, "bubbling up" the larger elements.

*Algorithm:*

*Step 1 – Set MIN to location 0*

*Step 2 – Search the minimum element in the list*

*Step 3 – Swap the value at location MIN with the value at location i.*

*Step 4 – Increment MIN to point to next element*

*Step 5 – Repeat steps 2-4 until the list is sorted.*

### – Insertion Sort

Builds the sorted portion of the array one element at a time by inserting each element in its correct position.

*Algorithm:*

*Step 1 – Start from the second element (index 1) of the array.*

*Step 2 – Set the key as the current element's value.*

*Step 3 – Compare the key with the elements before it.*

*Step 4 – Compare the key with the elements before it.*

*Step 5 – Insert the key into the appropriate position.*

*Step 6 – Repeat steps 2-5 for all elements.*



## - Merge Sort

Divides the array into smaller subarrays, sorts them, and then merges them back together in order.

*Algorithm:*

*Step 1 – Divide the array into two halves.*

*Step 2 – Divide the array into two halves*

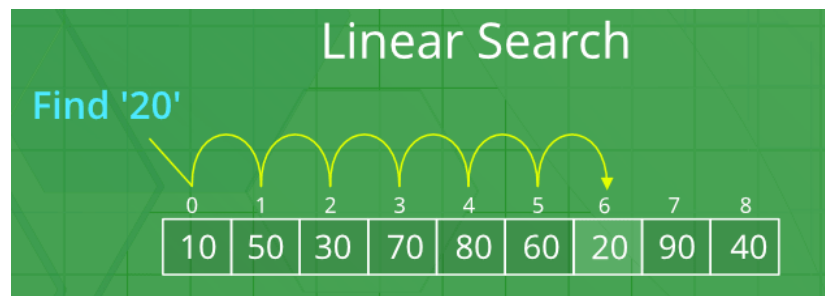
*Step 3 – Merge the two sorted halves into a single sorted array.*

*Step 4 – Repeat the process until the array is completely sorted.*

## - Searching Algorithms

### - Linear search Algorithm

Iterates through each element, comparing the target value until found.



*Algorithm:*

*Step 1: Set  $i$  to 1*

*Step 2: if  $i > n$  then go to step 7*

*Step 3: if  $A[i] = x$  then go to step 6*

*Step 4: Set  $i$  to  $i + 1$*

*Step 5: Go to Step 2*

*Step 6: Print Element  $x$  Found at index  $i$  and go to step 8*

*Step 7: Print element not found*

*Step 8: Exit*

### - Binary Search Algorithm

Efficiently searches sorted data by repeatedly dividing the search space in half.

| Binary Search                        |     |   |   |    |     |     |          |     |    |     |
|--------------------------------------|-----|---|---|----|-----|-----|----------|-----|----|-----|
|                                      | 0   | 1 | 2 | 3  | 4   | 5   | 6        | 7   | 8  | 9   |
| Search 23                            | 2   | 5 | 8 | 12 | 16  | 23  | 38       | 56  | 72 | 91  |
|                                      | L=0 |   |   |    | M=4 |     |          |     |    | H=9 |
| 23 > 16<br>take 2 <sup>nd</sup> half | 2   | 5 | 8 | 12 | 16  | 23  | 38       | 56  | 72 | 91  |
|                                      |     |   |   |    |     | L=5 |          | M=7 |    | H=9 |
| 23 < 56<br>take 1 <sup>st</sup> half | 2   | 5 | 8 | 12 | 16  | 23  | 38       | 56  | 72 | 91  |
|                                      |     |   |   |    |     |     | L=5, M=5 | H=6 |    |     |
| Found 23,<br>Return 5                | 2   | 5 | 8 | 12 | 16  | 23  | 38       | 56  | 72 | 91  |

*Algorithm:*

*Step 1: set beg = lower\_bound,*

*end=upper\_bound, pos = - 1*

*Step 2: repeat steps 3 and 4 while beg <=end*

*Step 3: set mid = (beg + end)/2*

*Step 4: if a[mid] = val*

*set pos = mid*

*print pos*

*go to step 6*

*else if a[mid] > val*

*set end = mid - 1*

*else*

*set beg = mid + 1*

*[end of if]*

*[end of loop]*

*Step 5: if pos = -1*

*print "value is not present in the array"*

*[end of if]*

*Step 6: exit*

## What is File Handling?

File handling involves managing data by reading from and writing to files on a computer's storage, enabling persistent data storage and retrieval.

## Why File Handling required?

- **Data Persistence:** Ensures data is saved beyond program execution.
- **Data Sharing:** Enables communication between programs and users.
- **Configuration:** Stores settings and preferences.
- **Storage:** Efficiently handles large datasets and files.
- **Backup:** Provides data backup and recovery.
- **Logging:** Records activities and maintains logs.
- **Manipulation:** Allows external data modification.
- **Integrity:** Ensures data security and integrity.

## File Handling in Java

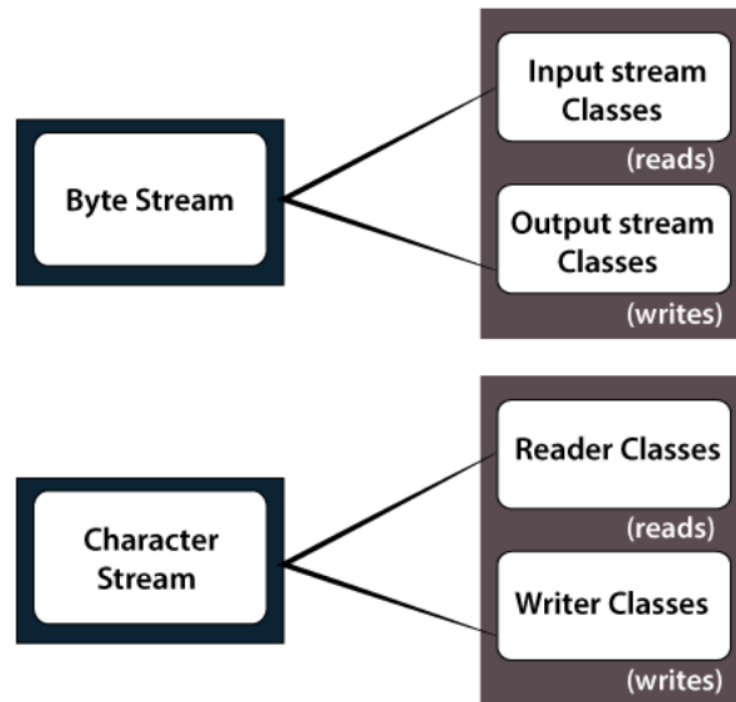
- **File as Abstract Data Type:** Represents a named location for storing related information.
- **File Operations:** Includes creating, obtaining information, writing, reading, and deleting files.
- **Effective Data Management:** Enables efficient data manipulation and storage within Java programs.

## What is Stream in Java?

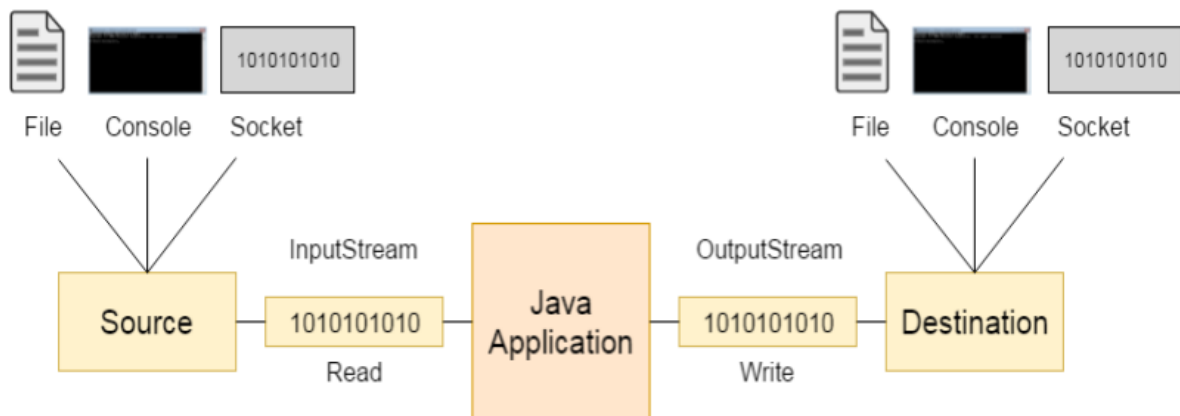
A stream is a data sequence in Java used for reading from and writing to various I/O sources, simplifying I/O operations and enabling data transformation

## Types of Stream

- **Byte Stream**
  - Deal with binary data.
  - Used for reading and writing raw bytes.
  - Suitable for handling all kinds of files.
- **Character Stream.**
  - Handle character-based data.
  - Used for reading and writing characters.
  - Provide character encoding support for text files.



## Input Stream vs Output Stream



## Input Stream in Java

- Java's `InputStream` is the base class for input streams.
- Used for reading data from input devices like keyboards or networks.
- Subclasses provide specialized methods for actual data reading:

- AudioInputStream
- ByteArrayInputStream
- FileInputStream
- FilterInputStream
- StringBufferInputStream
- ObjectInputStream

### Methods of Input Stream

| S No. | Method               | Description   |
|-------|----------------------|---|
| 1     | read()               | Reads one byte of data from the input stream.                                     |
| 2     | read(byte[] array)() | Reads byte from the stream and stores that byte in the specified array.           |
| 3     | mark()               | It marks the position in the input stream until the data has been read.           |
| 4     | available()          | Returns the number of bytes available in the input stream.                        |
| 5     | markSupported()      | It checks if the mark() method and the reset() method is supported in the stream. |
| 6     | reset()              | Returns the control to the point where the mark was set inside the stream.        |
| 7     | skip()               | Skips and removes a particular number of bytes from the input stream.             |
| 8     | close()              | Closes the input stream.  |

## Output Stream in Java

- Used to write data to output devices like monitors and files.
- OutputStream is an abstract superclass for output streams.
- Subclasses handle actual data writing.
- Subclasses include ByteArrayOutputStream, FileOutputStream, etc.

# Methods of Output Stream

| S. No. | Method              | Description  |
|--------|---------------------|--|
| 1.     | write()             | Writes the specified byte to the output stream.                              |
| 2.     | write(byte[] array) | Writes the bytes which are inside a specific array to the output stream.     |
| 3.     | close()             | Closes the output stream.  |
| 4.     | flush()             | Forces to write all the data present in an output stream to the destination. |

# Java File Class Methods

| S.No. | Method          | Return Type | Description   |
|-------|-----------------|-------------|---|
| 1.    | canRead()       | Boolean     | The <b>canRead()</b> method is used to check whether we can read the data of the file or not.     |
| 2.    | createNewFile() | Boolean     | The <b>createNewFile()</b> method is used to create a new empty file.                             |
| 3.    | canWrite()      | Boolean     | The <b>canWrite()</b> method is used to check whether we can write the data into the file or not. |
| 4.    | exists()        | Boolean     | The <b>exists()</b> method is used to check whether the specified file is present or not.         |

|     |                                |          |  |
|-----|--------------------------------|----------|--|
| 5.  | <code>delete()</code>          | Boolean  | The <b><code>delete()</code></b> method is used to delete a file.                                      |
| 6.  | <code>getName()</code>         | String   | The <b><code>getName()</code></b> method is used to find the file name.                                |
| 7.  | <code>getAbsolutePath()</code> | String   | The <b><code>getAbsolutePath()</code></b> method is used to get the absolute pathname of the file.     |
| 8.  | <code>length()</code>          | Long     | The <b><code>length()</code></b> method is used to get the size of the file in bytes.                  |
| 9.  | <code>list()</code>            | String[] | The <b><code>list()</code></b> method is used to get an array of the files available in the directory. |
| 10. | <code>mkdir()</code>           | Boolean  | The <b><code>mkdir()</code></b> method is used for creating a new directory.                           |

## What is System Software?

Software that manages hardware, provides a platform for apps, and enables communication; includes OS, drivers, utilities, and libraries for efficient computer operation.

## Features of System Software

1. Hardware Management: Efficiently manages computer hardware resources.
2. Platform Creation: Provides a consistent platform for applications.
3. Resource Allocation: Allocates resources like memory and CPU time.
4. User Interface: Offers interfaces for user interaction.
5. Device Drivers: Enables hardware-software communication.
6. Security: Implements data and system security.
7. Maintenance: Provides tools for system upkeep.
8. Error Handling: Manages errors and exceptions.
9. Multitasking: Supports running multiple applications.
10. Compatibility: Ensures hardware compatibility.

## Types of System Software

- **Operating System:** Manages hardware, software, and resources; provides user interface and multitasking support.  
Examples include Windows, macOS, Linux.
- **Programming Language Translators:** Convert high-level programming code into machine-readable instructions.  
Compilers (e.g., GCC), interpreters (e.g., Python), assemblers.
- **Device Drivers:** Enable communication between hardware devices and the operating system.  
NVIDIA graphics driver, printer driver, USB driver.
- **Firmware Software:** Embedded software in hardware devices for basic functionality.  
BIOS/UEFI firmware in a computer, firmware in routers.
- **BIOS/UEFI:** Firmware that initializes hardware components during system startup.  
BIOS (Basic Input/Output System) or UEFI (Unified Extensible Firmware Interface) in computers for hardware initialization.

## Explain Case Study

A detailed exploration of a real-world problem to practice programming skills, involving problem analysis, solution design, coding, testing, and evaluation.



## What are the advantages of case studies?

- Practical Application: Applies theory to real-world situations.
- Problem-Solving: Enhances critical thinking and problem-solving skills.
- Hands-on Experience: Offers practical coding practice.
- Contextual Learning: Demonstrates how concepts apply in various contexts.
- Decision Making: Involves making decisions, improving decision-making skills.
- Team Collaboration: Mimics real project dynamics, enhancing teamwork.
- Error Analysis: Identifies and learns from common mistakes.
- Learning from Failure: Failing provides valuable learning opportunities.
- Portfolio Building: Enhances programming portfolio.
- Career Preparation: Equips learners for real job challenges

## What is Algorithm Analysis?

- Importance: Analyzing algorithms estimates their resource needs for problem solving.
- Variable Inputs: Algorithms handle diverse input lengths; analysis evaluates time and space resources.
- Efficiency Metrics: Efficiency is gauged by time complexity (input vs. steps) and space complexity (memory usage), quantifying performance.

## Asymptotic Notation and Analysis

- Purpose: Analyzes algorithm performance as input size increases.
- Simplification: Focuses on significant resource consumption parts.
- Notations: Big O (upper bound), Omega (lower bound), Theta (tight bound).
- Scalability: Helps choose efficient solutions for varying input sizes.

Join Discord: <https://discord.gg/VRmAjYk5qs>



Thank you, everyone. Here's a complete short note. Please note that we haven't included the UI part as we're covering that in another subject. If you need this part, please message