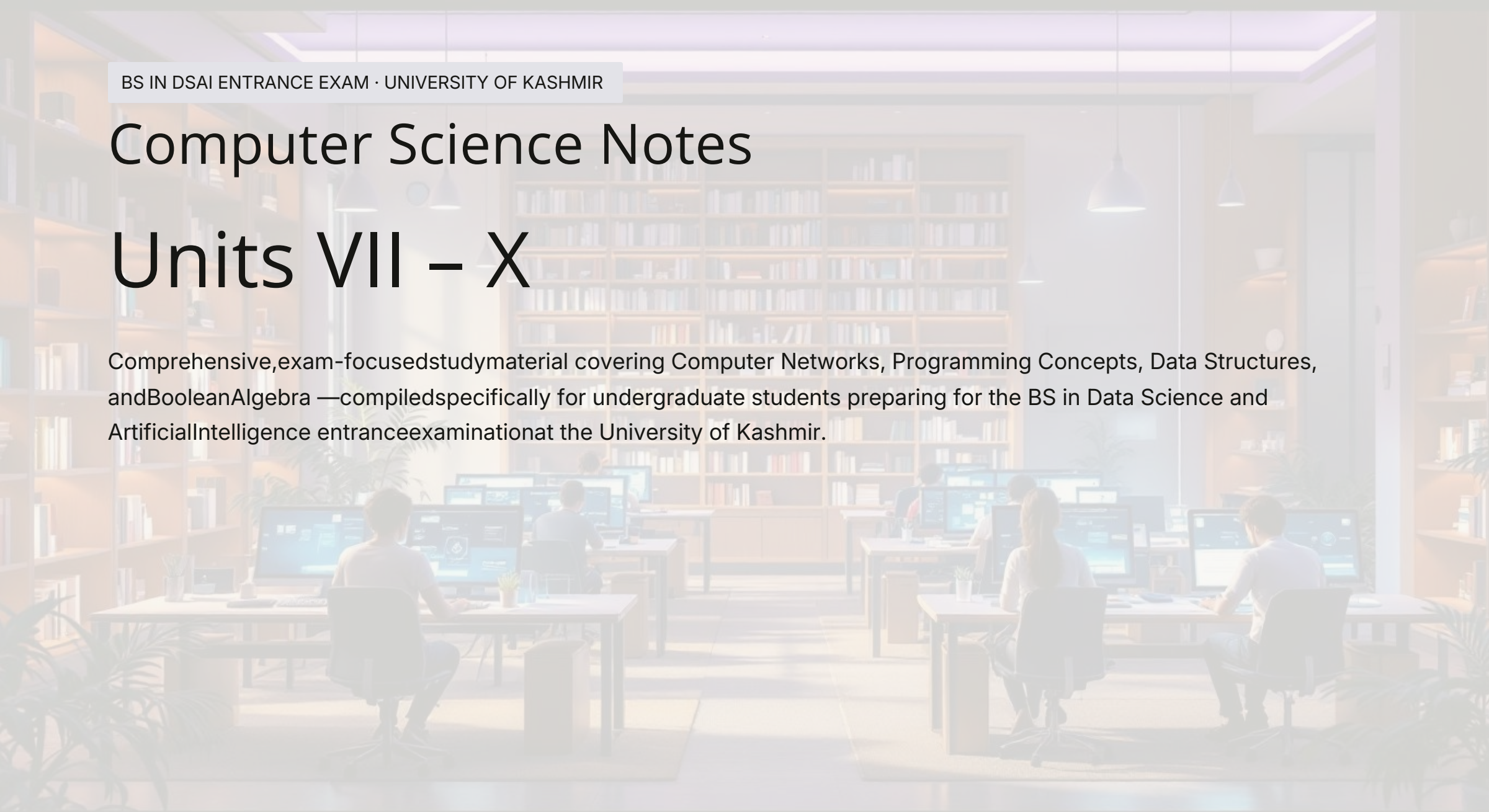
BS IN DSAI ENTRANCE EXAM · UNIVERSITY OF KASHMIR

Computer Science Notes

Units VII – X

Comprehensive,exam-focusedstudymaterial covering Computer Networks, Programming Concepts, Data Structures, andBooleanAlgebra —compiledspecifically for undergraduate students preparing for the BS in Data Science and ArtificialIntelligence entranceexaminationat the University of Kashmir.



AUTHOR PAGE

About This Document

Compiled By

Syed Sajid Ul Haq

Educator, Python developer, and prompt engineering technician

Publisher

CrescaDemy

Academic Learning Platform

Website

crescademy.netlify.app

Free study resources & notes

Syllabus

BS in DSAI Entrance Exam

University of Kashmir — Latest Syllabus

> **Note for Students:** These notes are structured for both conceptual understanding and rapid revision. Key terms are **bolded** for emphasis. Diagrams, tables, and comparison charts are included to help you visualize abstract concepts. Use these notes alongside your textbooks and practice problems for best results.

Each unit is organized with clear definitions, properties, examples, and visual aids. Pay special attention to comparison tables and diagrams — these are frequently tested in entrance examinations. The content is aligned with the latest syllabus prescribed by the University of Kashmir for the BS in Data Science and Artificial Intelligence programme.

UNIT VII

Computer Networks — Foundations

A**computernetwork** is a setof interconnected devices(computers, servers,routers, switches) that can communicate andshare resources with each other. Networks are essential inmoderncomputing— theyenable resource sharing, communication, remote access, and distributed processing. Without networks, each computer would operate in isolation, making collaboration and data exchange extremely difficult.

Need for Computer Networks

- Resource sharing (printers, files, storage)
- Communication (email, messaging, video calls)
- Remote access to systems and data
- Centralized data management and backup
- Cost reduction through shared infrastructure

Advantages of Computer Networks

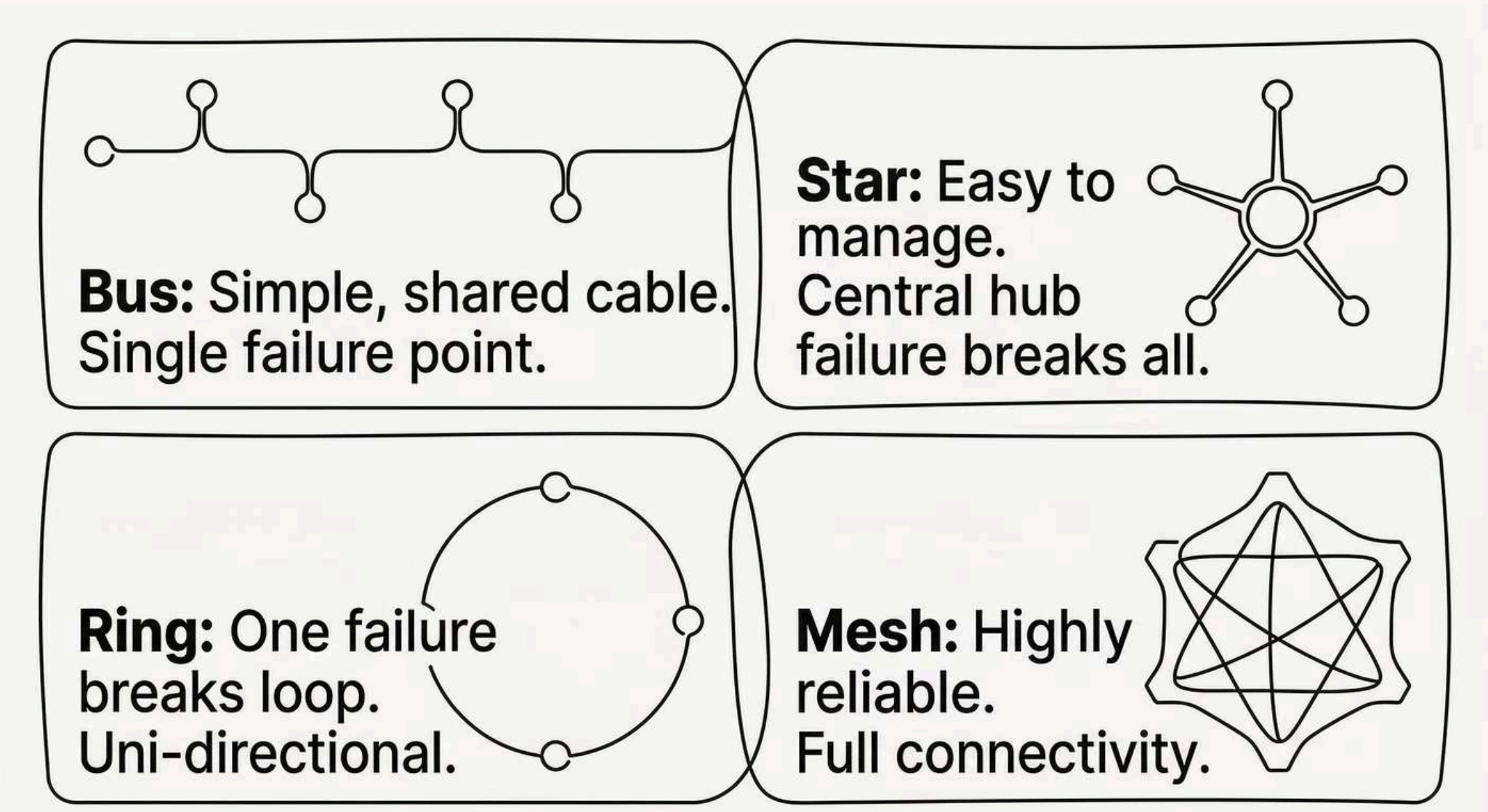
- High reliability through redundancy
- Scalability — easy to add new nodes
- Improved collaboration and teamwork
- Centralized software updates and management
- Access to distributed databases and cloud ser vices

Types of Networks

Type	Full Form	Coverage Area	Example
LAN	Local Area Network	Up to 1 km (room, building, c ampus)	Office network, school lab
MAN	Metropolitan Area Network	Up to 50 km (city-wide)	Cable TV network, city WiFi
WAN	Wide Area Network	Countries and continents	Internet, banking networks
PAN	Personal Area Network	Up to 10 meters (personal devices)	Bluetooth, USB tethering

Network Topologies

A **network topology** defines the physical or logical arrangement of devices in a network. The choice of topology affects cost, performance, fault tolerance, and ease of installation.



Bus Topology

All devices share a single communication line (backbone). Simple and cheap, but a cable break disables the entire network.

Star Topology

All devices connect to a central hub or switch. Easy to install and manage. If the hub fails, the entire network goes down.

Ring Topology

Devices form a closed loop. Data travels in one direction. A single node failure can break the entire ring.

Mesh Topology

Every device connects to every other device. Highly fault-tolerant but expensive. Used in critical infrastructure.

UNIT VII — CONTINUED

IP Addressing, Protocols & the Web

IP Addressing: IPv4 vs IPv6

An **IP (Internet Protocol) address** is a unique numerical label assigned to every device on a network. It identifies the device and its location, enabling proper routing of data packets.

Feature	IPv4	IPv6
Address Length	32-bit	128-bit
Address Format	Dotted decimal (e.g., 192.168.1.1)	Hexadecimal (e.g., 2001:0db8::1)
Total Addresses	~4.3 billion	~340 undecillion
Header Size	20–60 bytes	Fixed 40 bytes
Secur it y	Optional (IPSec add-on)	Built-in IPSec support
Configuration	Manual or DHCP	Auto-configuration supported

Public vs Private IP Addresses: A **public IP** is routable on the internet and uniquely identifies your network globally. A **private IP** is used within a local network (e.g., 192.168.x.x, 10.x.x.x) and is not visible on the internet.

Network Protocols

HTTP / HTTPS

HyperText Transfer Protocol — used for transferring web pages. **HTTPS** is the secure version using TLS/SSL encryption (port 443 vs port 80).

FTP

File Transfer Protocol — used for transferring files between client and server. Supports authentication and directory browsing.

SMTP

Simple Mail Transfer Protocol — used for sending emails from client to server or between mail servers (port 25 or 587).

TCP/IP

Transmission Control Protocol / Internet Protocol — the foundational suite of the internet. TCP ensures reliable, ordered delivery; IP handles addressing and routing.

Internet vs World Wide Web

Internet

The **Internet** is the physical infrastructure — a global network of interconnected computers and cables. It is the underlying hardware and protocols that enable communication.

- Infrastructure layer
- Uses TCP/IP protocols
- Includes email, FTP, VoIP, etc.

World Wide Web

The **WWW** is a service that runs *on top of* the internet. It is a collection of interconnected documents and resources accessed via URLs and hyperlinks using HTTP/HTTPS.

- Application layer service
- Uses HTTP/HTTPS protocols
- Accessed via web browsers

Web browsers (Chrome, Firefox, Safari, Edge) are software applications that fetch, interpret, and display web pages. They send HTTP requests to servers, receive HTML/CSS/JS responses, and render them visually. Modern communication tools include email clients, instant messaging (WhatsApp, Telegram), video conferencing (Zoom, Meet), and collaborative platforms (Slack, Teams).

UNIT VIII

Programming Concepts — Algorithms & Flowcharts

An**algorithm** isa finite, well-defined sequenceofinstructions designed to solve aspecific problem orperform a computation. Every algorithm must be **unambiguous**,**finite**,**effective**, and must produce a correct output for every valid input. Algorithms are the foundation of all computer programs — before writing code, a programmer designs an algorithm to solve the problem logically.

1

Input

Zero or more well-defined inputs are accepted from the user or another source.

2

Output

At least one well-defined output is produced as the result of execution.

3

Definiteness

Each step must be clear and unambiguous — no room for interpretation.

4

Finiteness

The algorithm must terminate after a finite number of steps.

5

Effectiveness

Each step must be basic enough to be carried out precisely in finite time.

Example: Algorithm to Find the Largest of Three Numbers

01

START

02

Read three numbers: A, B, C

03

If A ≥ B and A ≥ C, then Largest = A

04

Else if B ≥ C, then Largest = B

05

Else Largest = C

06

Print Largest

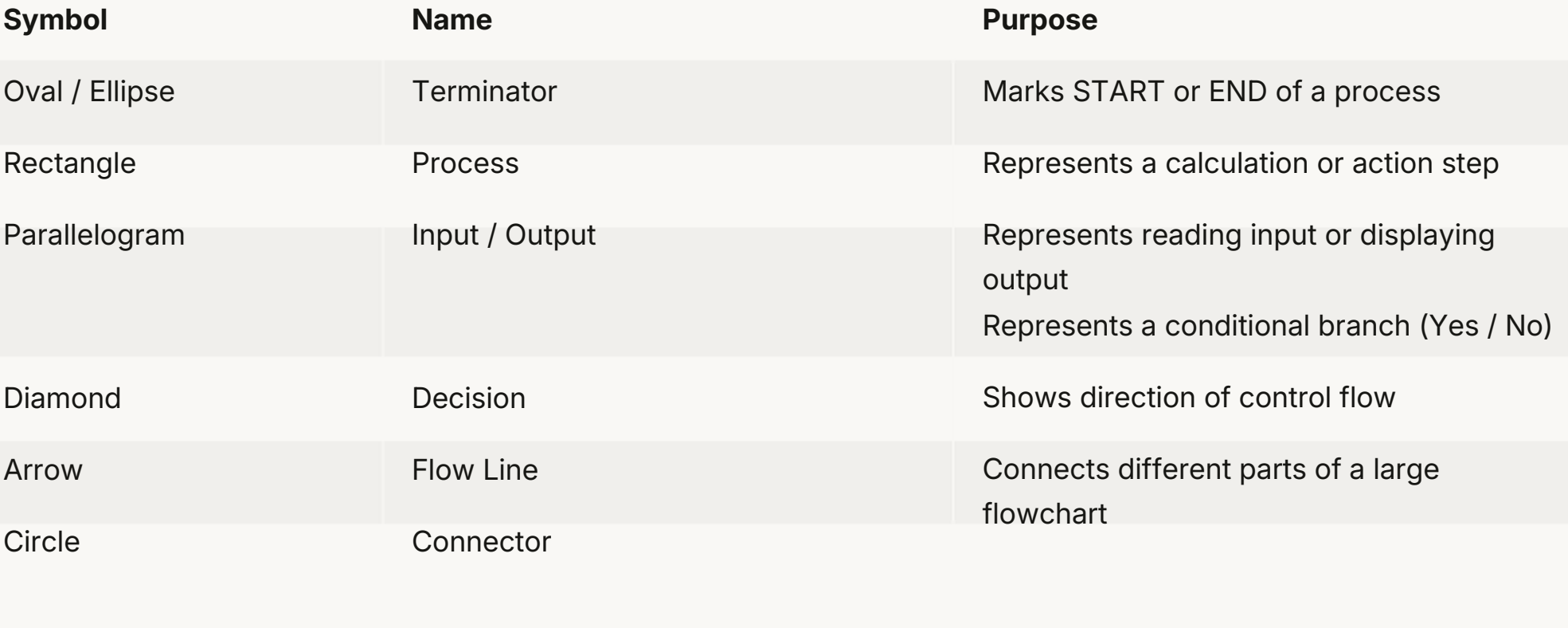
07

STOP

Flowcharts

A **flowchart** is a graphical representation of an algorithm using standard symbols. It helps visualize the flow of control and logic before coding begins.

Symbol	Name	Purpose
Oval / Ellipse	Terminator	Marks START or END of a process
Rectangle	Process	Represents a calculation or action step
Parallelogram	Input / Output	Represents reading input or displaying output
Diamond	Decision	Represents a conditional branch (Yes / No)
Arrow	Flow Line	Shows direction of control flow
Circle	Connector	Connects different parts of a large flowchart



The flowchart above illustrates the decision-making process for finding the largest of three numbers. Each diamond represents a conditional check, and arrows direct the flow based on the outcome (Yes/No). Flowcharts are invaluable for planning program logic before writing actual code.

UNIT VIII — CONTINUED

Data Types, Control Structures & OOP

Data Types

Primitive Data Types

Basic, built-in types that store a single value directly.

- **int** — Integer values (e.g., 5, -3)
- **float** — Decimal values (e.g., 3.14)
- **char** — Single character (e.g., 'A')
- **boolean** — True or False
- **double** — High-precision decimal

Non-Primitive Data Types

Derived from primitive types; can store multiple values.

- **Array** — Fixed-size collection of same-type elements
- **String** — Sequence of characters
- **List** — Dynamic, ordered collection
- **Stack / Queue** — Specialized linear structures
- **Tree / Graph** — Non-linear structures

Control Structures

Sequence

Statements execute one after another in order. Default flow of any program.

Selection

Executes different blocks based on a condition. Includes **if-else** and **switch-case**.

Iteration

Repeats a block while a condition holds. Includes **for**, **while**, and **do-while** loops.

Functions & Recursion

A **function** is a named block of code that performs a specific task and can be called from other parts of a program. Functions promote code reusability, modularity, and easier debugging. Types include **built-in functions** (e.g., len(), print()) and **user-defined functions**.

Recursion is a technique where a function calls itself to solve a smaller instance of the same problem. Every recursive function must have a **base case** (termination condition) to prevent infinite recursion. Example: `factorial(n) = n × factorial(n-1)`, with base case `factorial(0) = 1`.

Object-Oriented Programming (OOP)

OOP is a programming paradigm based on the concept of **objects**, which contain data (attributes) and behavior (methods). It models real-world entities and promotes code organization, reuse, and maintainability.

Encapsulation

Bundling data and methods that operate on that data within a single unit (class). Restricts direct access to some components using access modifiers (private, public).

Inheritance

A child class acquires properties and behaviors of a parent class. Promotes code reuse and hierarchical classification. Example: Dog inherits from Animal.

Polymorphism

The ability of an object to take many forms. A method behaves differently based on the object calling it. Achieved through **method overriding** and **method overloading**.

Abstraction

Hiding complex implementation details and showing only essential features. Achieved using **abstract classes** and **interfaces**. Example: A user knows *what* a function does, not *how*.

UNIT IX

Data Structures — Introduction, Lists & Strings

A **data structure** is a systematic way of organizing and storing data in a computer so that it can be accessed and modified efficiently. The choice of data structure directly impacts the performance of algorithms — a well-chosen structure can reduce time complexity from $O(n^2)$ to $O(n \log n)$ or even $O(1)$.

Why Data Structures Matter

- Efficient data storage and retrieval
- Optimized algorithm performance
- Better memory management
- Foundation for advanced algorithms

Types of Data Structures

- **Linear:** Array, List, Stack, Queue
- **Non-Linear:** Tree, Graph
- **Homogeneous:** All elements same type
- **Heterogeneous:** Mixed element types

Lists

A **list** is an ordered, mutable (changeable) collection of elements. In Python, lists are dynamic arrays that can store elements of different types. Lists support indexing, slicing, and a variety of built-in operations.

List Operations

- **Append:** Add element at end — `list.append(x)`
- **Insert:** Add at position — `list.insert(i, x)`
- **Remove:** Delete by value — `list.remove(x)`
- **Pop:** Remove by index — `list.pop(i)`
- **Sort:** Arrange in order — `list.sort()`
- **Slice:** Access range — `list[1:4]`

List Properties

- Ordered — maintains insertion order
- Mutable — elements can be changed
- Allows duplicate values
- Indexed — accessed by position (0-based)
- Dynamic size — grows/shrinks automatically

Strings

A **string** is a sequence of characters enclosed in quotes. In most languages, strings are **immutable** — once created, they cannot be changed. Strings support indexing, slicing, concatenation, and many built-in methods.

→ ### String Operations

Concatenation (`+`), **Repetition** (`*`), **Slicing** (`s[1:5]`), **Length** (`len(s)`), **Membership** (`in`)

→ ### Common String Methods

`.upper()`, `.lower()`, `.split()`, `.replace()`, `.strip()`, `.find()`, `.count()`

→ ### String Properties

Immutable, ordered, indexed, supports negative indexing (`s[-1]` = last character), supports escape sequences (`\n`, `\t`)

Stack, Queue & Linked List

Stack — LIFO Structure

A **stack** is a linear data structure that follows the **Last In, First Out (LIFO)** principle — the last element added is the first one removed. Think of a stack of plates: you can only add or remove from the top.

Stack Operations

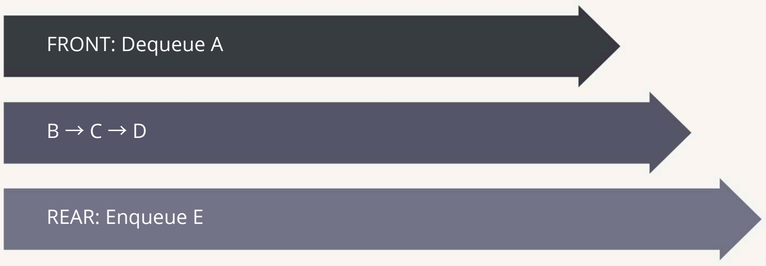
- **Push:** Add element to top
- **Pop:** Remove from top
- **Peek:** View top element
- **isEmpty:** Check if empty
- **isFull:** Check if full (fixed size)

Real-Life Examples

- Browser back/forward navigation
- Undo/Redo in text editors
- Function call stack in recursion
- Evaluation of arithmetic expressions

Queue — FIFO Structure

A **queue** is a linear data structure that follows the **First In, First Out (FIFO)** principle — the first element added is the first one removed. Think of a ticket counter queue: the first person in line is served first.



Simple Queue

Basic FIFO queue. Elements added at rear, removed from front. Limited by fixed size.

Circular Queue

Last position connects back to first, forming a circle. Solves memory wastage in simple queue.

Priority Queue

Elements are dequeued based on priority, not order. Higher priority elements are served first.

Double-Ended Queue

Deque — allows insertion and deletion at both ends (front and rear).

Linked List

A **linked list** is a linear data structure where each element (called a **node**) contains data and a pointer to the next node. Unlike arrays, linked lists do not require contiguous memory allocation.

Node Structure

Each node has two parts:

- **Data Field:** Stores the actual value
- **Next Pointer:** Stores address of next node

Last node points to **NULL** (end of list).

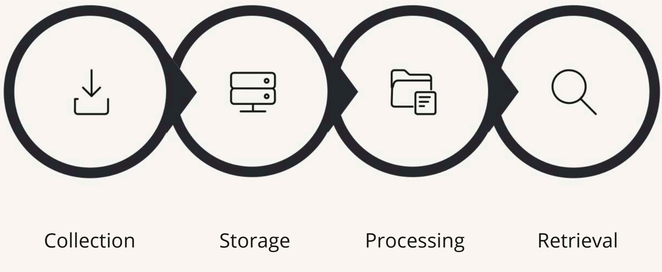
Types of Linked Lists

- **Singly Linked:** One-way traversal
- **Doubly Linked:** Forward and backward traversal
- **Circular Linked:** Last node points back to head

Advantages vs Disadvantages

- Dynamic size, no memory waste
- Efficient insert/delete at any position
- No random access (must traverse)
- Extra memory for pointers

Data Handling Process



The data handling process is a systematic pipeline that transforms raw data into meaningful information. **Collection** involves gathering data from sensors, forms, or databases. **Storage** organizes data in structured formats (files, databases). **Processing** applies algorithms and computations to extract insights. Finally, **retrieval** presents processed results to users through reports, dashboards, or interfaces.

UNIT X

Boolean Algebra — Operations & Logic Gates

Boolean Algebra is a branchof mathematics developedby George Boolein1854. It dealswith binaryvariables (0and 1, orFALSE andTRUE) and logical operations. Booleanalgebra is the mathematical foundation of alldigitalcircuits and computer logic — every operation your computer performs is ultimately reduced to Boolean expressions.

Basic Boolean Operations

AND Operation (·)

Output is **1** only when *all* inputs are 1.

A · B = 1 only if A=1 **and** B=1

Otherwise output = 0

OR Operation (+)

Output is **1** when *at least one* input is 1.

A + B = 0 only if A=0 **and** B=0

Otherwise output = 1

NOT Operation (')

Output is the **complement** (inverse) of input.

A' = 1 if A=0

A' = 0 if A=1

XOR Operation (⊕)

Output is **1** when inputs are *different*.

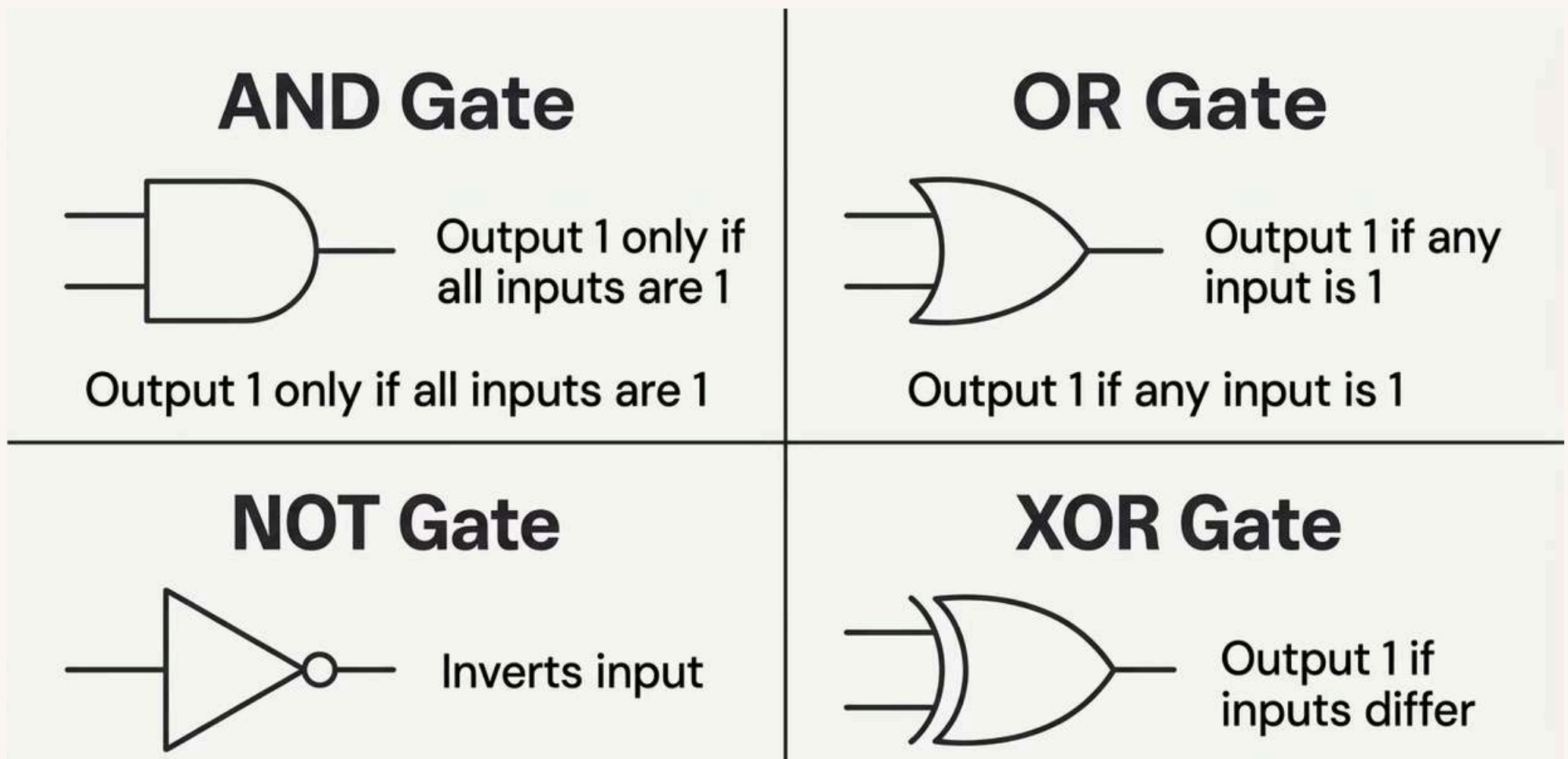
A ⊕ B = 1 if A ≠ B

A ⊕ B = 0 if A = B

Logic Gates & Truth Tables

A **logic gate** is a physical electronic device that implements a Boolean function. Gates are the building blocks of all digital circuits — processors, memory chips, and all computing hardware are made of billions of interconnected logic gates.

A	B	AND (A·B)	OR (A+B)	XOR (A⊕B)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



NOT Gate Truth Table

A	A' (NOT A)
0	1
1	0

Gate Symbols at a Glance

- **AND:** D-shaped body, flat input side
- **OR:** Curved input side, pointed output
- **NOT:** Triangle with small circle (bubble) at output
- **XOR:** OR gate with additional curved line at input

The small circle (bubble) on any gate indicates **inversion** (NOT operation).

UNIT X — CONTINUED

Boolean Laws & Applications

Boolean lawsare fundamental identities thatgovernthe manipulationand simplification of Boolean expressions. These laws are essential for designing efficient digital circuits and for simplifying logical conditions in programming.

Key Boolean Laws

Law	AND Form	OR Form
Identit y	A · 1 = A	A + 0 = A
Null (Domination)	A · 0 = 0	A + 1 = 1
Idempotent	A · A = A	A + A = A
Complement	A · A' = 0	A + A' = 1
Involution	(A')' = A	(A')' = A
Absorption	A · (A + B) = A	A + (A · B) = A

De Morgan's Laws

De Morgan's Laws are among the most important in Boolean algebra. They describe how to distribute a NOT operation over AND or OR expressions.

De Morgan's First Law

The complement of a product equals the sum of complements:

(A · B)' = A' + B'

The NOT of (A AND B) = (NOT A) OR (NOT B)

De Morgan's Second Law

The complement of a sum equals the product of complements:

(A + B)' = A' · B'

The NOT of (A OR B) = (NOT A) AND (NOT B)

Exam Tip: De Morgan's Laws are frequently tested. Remember the mnemonic: **"Break the bar, change the sign"** — when you apply NOT over an expression, break the overline and swap AND with OR (or vice versa).

Applications of Boolean Algebra

Digital Circuit Design

Boolean algebra is used to design and simplify logic circuits in processors, memory units, and all digital hardware. Engineers use Karnaugh Maps (K-Maps) to minimize Boolean expressions and reduce circuit complexity.

Programming Logic

Every conditional statement (`if`, `while`, `for`) in programming uses Boolean expressions. Understanding Boolean laws helps write efficient, bug-free conditional logic.

Database Queries

SQL queries use Boolean logic (AND, OR, NOT) to filter and retrieve data. Search engines also rely on Boolean operators for advanced search functionality.

AI & Machine Learning

Boolean logic underpins decision trees, rule-based systems, and neural network activation functions. It is foundational to automated reasoning and expert systems in AI.

> **Revision Summary — Units VII to X:** Computer Networks (types, topologies, protocols, IP addressing) → Programming Concepts (algorithms, flowcharts, data types, OOP) → Data Structures (stack, queue, linked list) → Boolean Algebra (operations, gates, laws, De Morgan's). Master these fundamentals and you have a strong foundation for the BS in DSAI entrance exam.