# SQL

**Basic Commands**

* **Show all databases**:  
  SHOW DATABASES;
* **Switch to a database**:  
  USE db\_name;
* **Show all tables in the database**:  
  SHOW TABLES;
* **Describe a table structure**:  
  DESCRIBE table\_name;

**Common Queries**

1. **Why use LIMIT and OFFSET?**
   * **LIMIT**: Efficient resource usage by restricting the number of results.
   * **OFFSET**: Enables pagination by skipping a specified number of rows.
2. **Get 20 rows of name and rankscore starting from the 41st row in the movies table**:

sql

Copy code

SELECT name, rankscore

FROM movies

LIMIT 20 OFFSET 40;

1. **List actors whose first names start with 'L' but do not start with 'Li'**:

sql

Copy code

SELECT first\_name, last\_name

FROM actors

WHERE first\_name LIKE 'L%'

AND first\_name NOT LIKE 'Li%';

1. **List movies released after the year 2000 (without GROUP BY)**:

sql

Copy code

SELECT name, year

FROM movies

WHERE year > 2000;

1. **Find years with more than 20 movies having a rankscore greater than 9**:

sql

Copy code

SELECT year, COUNT(\*) AS year\_count

FROM movies

WHERE rankscore > 9

GROUP BY year

HAVING year\_count > 20;

**Key SQL Concepts**

* **Order of Execution**:
  1. GROUP BY: Groups rows with identical values in specified columns.
  2. Aggregate Functions (COUNT, SUM, etc.): Perform calculations on grouped data.
  3. HAVING: Filters grouped results based on conditions.
* **HAVING vs. WHERE**:
  1. WHERE: Applied to individual rows before grouping.
  2. HAVING: Applied to grouped results after aggregation.

**Advanced Queries**

1. **List names and genres of movies, limit results to 20 rows**:

sql

Copy code

SELECT m.name, g.genre

FROM movies m

JOIN movies\_genres g ON m.id = g.movie\_id

LIMIT 20;

1. **Include movies with no genres, limit results to 20 rows**:

sql

Copy code

SELECT m.name, g.genre

FROM movies m

LEFT JOIN movies\_genres g ON m.id = g.movie\_id

LIMIT 20;

1. **List first and last names of actors in 'Officer 444'**:

sql

Copy code

SELECT a.first\_name, a.last\_name

FROM actors a

JOIN roles r ON a.id = r.actor\_id

JOIN movies m ON m.id = r.movie\_id

WHERE m.name = 'Officer 444';

1. **List all actors in 'Schindler's List'**:

sql

Copy code

SELECT first\_name, last\_name

FROM actors

WHERE id IN (

SELECT actor\_id

FROM roles

WHERE movie\_id IN (

SELECT id

FROM movies

WHERE name = 'Schindler\'s List'

)

);

**Data Manipulation**

1. **Insert multiple movie records in one query**:

sql

Copy code

INSERT INTO movies (id, name, year, rankscore)

VALUES

(412321, 'Thor', 2011, 7),

(412322, 'Iron Man', 2008, 7.9),

(412323, 'Iron Man 2', 2010, 7);

1. **Copy rows from one table to another using a subquery**:

sql

Copy code

INSERT INTO target\_table (col1, col2)

SELECT col1, col2

FROM source\_table

WHERE condition;

1. **Update a movie's rankscore**:

sql

Copy code

UPDATE movies

SET rankscore = 9

WHERE id = 412321;

**Table Management**

1. **Create a new table**:

sql

Copy code

CREATE TABLE language (

id INT PRIMARY KEY,

lang VARCHAR(50) NOT NULL

);

1. **Enforce foreign key relationship**:

sql

Copy code

CREATE TABLE orders (

order\_id INT,

user\_id INT,

FOREIGN KEY (user\_id) REFERENCES users(id)

);

1. **Ensure price column is greater than 0**:

sql

Copy code

CREATE TABLE products (

id INT,

price DECIMAL(10, 2),

CHECK (price > 0)

);

1. **Add and modify a column**:

sql

Copy code

ALTER TABLE language ADD country VARCHAR(50);

ALTER TABLE language MODIFY country VARCHAR(50);

1. **Remove a column**:

sql

Copy code

ALTER TABLE language DROP country;

1. **Safely drop a table**:

sql

Copy code

DROP TABLE TableName IF EXISTS;

1. **Create an index on the name column**:

sql

Copy code

CREATE INDEX idx\_user\_name ON users(name);

# Operating System Notes

**Definition and Purpose**

An Operating System (OS) is software that acts as an intermediary between computer hardware and users. It manages hardware resources (CPU, memory, storage, etc.) and provides services for running application programs.

* **Purpose**: To ensure efficient and fair resource allocation, manage hardware, and provide a user-friendly interface.
* **Real-Life Example**: An OS is like a restaurant manager who coordinates between customers (users) and the kitchen staff (hardware). The manager ensures orders (tasks) are taken correctly, food is prepared (executed), and served efficiently while handling customer complaints (errors).

**Types of Operating Systems**

1. **Batch Operating System**: Processes similar jobs together as a batch to save time.
   * Example: Payroll processing systems where tasks are scheduled to run at night without user intervention.
2. **Time-Sharing Operating System**: Allows multiple users to use a computer simultaneously by quickly switching between them.
   * Example: University servers where multiple students access resources simultaneously.
3. **Distributed Operating System**: Spreads the load over multiple systems, making it appear as a single system to the user.
   * Example: Google Cloud services which distribute computations across various servers.
4. **Real-Time Operating System (RTOS)**: Guarantees responses to inputs within a specific time, critical for embedded systems.
   * Example: Anti-lock braking systems (ABS) in cars that must respond instantly to prevent accidents.

**Functions and Services Provided by an OS**

* **Process Management**
* **File System Management**
* **Device Management**
* **Security and Access Control**
* **User Interface**

**Process vs. Program**

| **Aspect** | **Process** | **Program** |
| --- | --- | --- |
| **Definition** | An active entity (running instance) | A passive entity (set of instructions) |
| **State** | Changes dynamically | Static |
| **Example** | A running browser | The browser executable file |
| **Real-Life Example** | A recipe being actively followed to cook a dish | A written recipe |

**Process States and State Transitions**

A process can be in one of the following states:

* **New**: Process is being created.
* **Ready**: Process is waiting to be assigned to the CPU.
* **Running**: Process instructions are being executed.
* **Waiting/Blocked**: Process is waiting for an event (like I/O completion).
* **Terminated**: Process has finished execution.

**Real-Life Example**: Imagine you're at a theme park:

* **New**: You just arrived.
* **Ready**: You are waiting in line for a ride.
* **Running**: You're on the ride.
* **Waiting**: You stopped for a snack break.
* **Terminated**: You leave the park.

**Process Control Block (PCB)**

A **Process Control Block (PCB)** is a data structure used by the OS to store information about a process, such as:

* Process ID
* Process state
* CPU registers
* Memory limits
* Open files

**Real-Life Example**: A PCB is like a student's profile in a school database containing details like roll number, name, current grade, and subjects enrolled.

**Context Switching**

**Definition**: The process of saving the state of a running process and loading the state of another process to ensure smooth multitasking.

**Real-Life Example**: Cooking multiple dishes and pausing one dish to work on another without losing progress.

**Security and Protection Goals:**

1. **Data Confidentiality**: Ensures only authorized access to data.
2. **Data Integrity**: Prevents unauthorized modifications.
3. **Data Availability**: Guarantees access when needed.
   * **Example**: Securing a bank vault.

**Encryption Basics:**

1. **Symmetric Encryption**:
   * **Same key** for encryption & decryption (e.g., AES).
   * **Use Case**: Data at rest.
2. **Asymmetric Encryption**:
   * **Public key** for encryption, **private key** for decryption (e.g., RSA).
   * **Use Case**: SSL, secure emails.

**Virtualization:**

* Creates virtual versions of OS, servers, or storage.
* **Purpose**: Optimize physical resources.
  + **Example**: Hotel rooms sharing the same facility.

**Types of Hypervisors**:

1. **Type 1 (Bare-Metal)**: Direct hardware installation (e.g., VMware ESXi).
   * Use: Data centers.
2. **Type 2 (Hosted)**: Runs on existing OS (e.g., VirtualBox).
   * Use: Development/testing.

**VMs vs. Containers:**

| **Aspect** | **VMs** | **Containers** |
| --- | --- | --- |
| **Definition** | Full OS on virtual HW | Isolated processes |
| **Resource Usage** | Higher | Minimal |
| **Boot Time** | Slower | Faster |
| **Examples** | VMware, VirtualBox | Docker, Kubernetes |

**Distributed Systems:**

* **Definition**: Independent computers functioning as one system.
* **Characteristics**:
  + Scalability, Fault Tolerance, Concurrency, Transparency.
  + **Example**: The Internet.

**Real-Time Operating Systems (RTOS):**

1. **Hard RTOS**: Guarantees strict deadlines (e.g., pacemakers).
2. **Soft RTOS**: Meets deadlines "most of the time" (e.g., multimedia systems).

**Linux vs. Windows OS:**

| **Aspect** | **Linux** | **Windows** |
| --- | --- | --- |
| **Nature** | Open-source | Proprietary |
| **Usage** | Development, servers | Home, enterprise |
| **Commands** | Command-line focused | GUI-focused |

# OOPs

### ****CLASSES AND OBJECTS****

**Definition:**

* A Class is a blueprint for creating objects. It defines properties (attributes) and behaviors (methods) that the objects created from it will have.
* An Object is an instance of a class, meaning it is a concrete representation of the blueprint.

**Real-Life Example:**

**Class**: Vehicle

* **Attributes**: Brand, model, color, speed.
* **Methods**: Start(), stop(), accelerate(), honk().

**Objects** (instances of the class):

* Object 1: A red **Toyota Corolla** that can start, stop, and honk.
* Object 2: A black **Honda Civic** with the same behaviors but different properties.

### ****FOUR PILLARS OF OOP****

#### **Encapsulation**

**Definition:**  
Encapsulation is the concept of wrapping data (attributes) and methods that operate on that data into a single unit (class) while restricting direct access to some of the object's components.

**Real-Life Example:**

* Imagine a Safe Box where valuables (attributes) are stored. You can only access the contents using a key (methods), not directly.
* A capsule which is mixed of several medicines. The medicines are hidden data to the end user.

#### **Abstraction**

**Definition:**  
Abstraction is the concept of hiding the internal implementation details and exposing only the essential features of an object.

**Real-Life Example:**

* A Coffee Machine exposes a simple interface (buttons for coffee, latte, etc.) but hides the internal brewing process.
* A driver will focus on the car functionality (Start/Stop -> Accelerate/Brake), he/she does not bother about how the accelerate/brake mechanism works internally. And this is how the abstraction works.

#### **Inheritance**

**Definition:**  
Inheritance is a mechanism where one class (child) derives attributes and behaviors from another class (parent).

**Real-Life Example:**

* A Library class has common attributes like books and methods for borrowing. A Digital Library class inherits from Library but adds digital features like e-books.
* The planet Earth and Mars inherit the superclass Solar System, and the Solar System inherits the Milky Way Galaxy. So Milky Way Galaxy is the top superclass for the classes Solar System, Earth, and Mars.

**Types of Inheritance:**

| **Type** | **Description** |
| --- | --- |
| Single Inheritance | A class inherits from a single parent class. |
| Multiple Inheritance | A class inherits from more than one parent class. |
| Multilevel Inheritance | A class is derived from a class, which is also derived from another class. |
| Hierarchical Inheritance | Multiple classes inherit from a single parent class. |
| Hybrid Inheritance | A combination of two or more types of inheritance. |

**Examples:**

* **Single Inheritance:** A Smart Lock system where a BasicLock class is extended by a SmartLock class to add features like remote access and notifications.
* **Multiple Inheritance:** A Flying Car inherits properties from both Car and Airplane. (Note: In Python, multiple inheritance is supported directly, but in Java and C++, it's done using interfaces or mixins.)
* **Multilevel Inheritance:** A University system where the University has a Department, which has a Professor.
* **Hierarchical Inheritance:** A Vehicle class has subclasses like Car and Bike, which inherit common attributes.
* **Hybrid Inheritance:** A Robot that can both Walk and Speak using properties inherited from both Walker and Speaker. (Note: Hybrid Inheritance is not directly supported in Java and C++ due to complexity and the diamond problem but can be achieved using interfaces or virtual inheritance.)

#### **Polymorphism**

**Definition:**  
Polymorphism allows methods to be used in different forms. It is achieved through method overloading and method overriding.

**Types of Polymorphism:**

1. **Compile-Time (Method Overloading):** Multiple methods with the same name but different parameters.
2. **Run-Time (Method Overriding):** A subclass provides a specific implementation of a method defined in its parent class.

**Real-Life Examples:**

* A MusicPlayer can play songs, but it may play different formats like MP3, WAV, or FLAC using the same method.
* A delivery person delivers items to the user: If it’s a postman, he will deliver the letters; if it’s a food delivery boy, he will deliver the foods. Polymorphism implements different ways for the delivery function.

**Definitions:**

* **Method Overloading (Compile-Time Polymorphism):** Defining multiple methods in the same class with the same name but different parameters.
* **Operator Overloading:** Allows redefining how operators work with user-defined types.  
  **Real-Life Example:** Overloading the + operator for a Vector class to add two vectors.
* **Method Overriding (Run-Time Polymorphism):** When a subclass redefines a method from its superclass with the same signature.

**Examples:**

* **Method Overloading:** A Smart Door can be opened using a key, a password, or a fingerprint.
* **Method Overriding:** Different types of doors (manual, automatic) have different ways to open.

### ****Constructor and Destructor****

**Definition:**

* **Constructor:** A special method called when an object is created to initialize it.
* **Destructor:** A method that is called automatically when an object is destroyed (not commonly used in Python and Java, but essential in C++).

**Real-Life Example:**

* **Constructor:** You set up your furniture and appliances when you move into a new apartment.
* **Destructor:** You clean up everything before leaving.

### ****Abstract Class and Interface****

**Definitions:**

* **Abstract Class:** A class that cannot be instantiated and may contain abstract methods (methods without implementation).
* **Interface:** A contract that defines methods that must be implemented by any class that uses it (Java-only concept).

**Real-Life Example:**

* A blueprint for a building:
  + The blueprint is an abstract class because it cannot be built directly.
  + The actual buildings (houses, apartments) are concrete implementations.

**Static Variables and Methods**

**Definition:**

* **Static Variables:** Class-level variables shared among all instances of the class.
* **Static Methods:** Methods that can be called without creating an instance of the class.

**Real-Life Example:**

* Imagine a School where the number of students is common to all classrooms.

**Final in Java**

**Definition:**

* **Final Variable:** Cannot change the value after initialization.
* **Final Method:** Cannot be overridden by subclasses.
* **Final Class:** Cannot be inherited.

**Real-Life Example:**

* Think of a National Monument: Once built, it cannot be modified or demolished.

**final vs finally**

* *Provide details or examples if needed.*

**Struct vs Class**

| **Struct** | **Class** | **Key Difference** |
| --- | --- | --- |
| Default members are **public**. | Default members are **private**. | Access specifier behavior differs. |
| Memory is **allocated on the stack**. | Memory is **allocated on the heap**. | Structs are typically used for lightweight objects. |
| **Does not support inheritance**. | **Supports inheritance**. | Structs are limited in OOP functionality compared to classes. |

**Friend Function**

**Definition:**

* A friend function is a function that has access to the private and protected members of a class.

**Real-Life Example:**

* Think of a trusted repairman who can enter your house (access private areas) but is not a family member (not part of the class).

**this Keyword**

**Definition:**

* The this keyword refers to the current object instance.

**Real-Life Example:**

* It's like saying, "I am referring to myself."

**Static Members**

**Definition:**

* Static members belong to the class, not individual instances.

**Real-Life Example:**

* A university where all students share a common registration number.

**Try-Catch Blocks**

**Definition:**

* Handles runtime errors gracefully without crashing the program.

**super Keyword**

**Definition:**

* The super keyword refers to the superclass and is used to access its members.

**Real-Life Example:**

* Think of inheriting a family heirloom and using it in your own house.

**Manual Memory Management**

**Definition:**

* In languages like C++, developers manually allocate (new) and deallocate (delete) memory. This gives more control but increases the risk of memory leaks if not handled properly.

**Real-Life Example:**

* Imagine renting a storage unit. You need to inform the storage company when you're done with it, or else you'll keep getting charged (memory leak).

**Garbage Collection**

**Definition:**

* Garbage Collection is an automatic memory management technique used by languages like Java and Python to reclaim memory that is no longer in use, preventing memory leaks.

**Real-Life Example:**

* Imagine cleaning your desk. If you don't throw away old, unnecessary papers (garbage), your desk will soon be cluttered and unusable. Similarly, garbage collection cleans up unused memory so your program can run efficiently.

**Multithreading and Concurrency**

**Definition:**

* Multithreading allows a program to run multiple threads simultaneously, enhancing performance, especially for I/O-bound or CPU-bound tasks.

**Real-Life Example:**

* Think of a restaurant kitchen where multiple chefs (threads) prepare different dishes at the same time.

**SOLID Principles**

| **Principle** | **Description** | **Real-Life Example** |
| --- | --- | --- |
| **S - Single Responsibility** | A class should have one and only one reason to change (one responsibility). | A Chef specializes only in cooking; handling billing and cleaning leads to inefficiency. |
| **O - Open/Closed Principle** | Classes should be open for extension but closed for modification. | A printer that supports additional formats (PDF, DOCX) without changing the original code. |
| **L - Liskov Substitution** | Subtypes must be substitutable for their base types without altering behavior. | If a bird can fly, its subclass sparrow should also fly. A subclass penguin violates LSP. |
| **I - Interface Segregation** | Clients should not be forced to depend on interfaces they do not use. | A smartphone supports many features, but a basic phone implements only calling/messaging. |
| **D - Dependency Inversion** | High-level modules should not depend on low-level modules but on abstractions. | A remote control should work with any TV, not just a specific model. |

**Common Design Patterns**

| **Pattern** | **Description** | **Real-Life Example** |
| --- | --- | --- |
| **Singleton** | Ensures that a class has only one instance and provides a global access point. | A government department where only one head can be appointed at a time. |
| **Factory** | Used to create objects without specifying the exact class. | A car factory producing different cars (SUV, Sedan, etc.) based on demand. |
| **Observer** | Notifies multiple objects about changes in the state of the subject. | A newsletter subscription notifying users of updates. |

# Linux

**Introduction**

Linux is a UNIX-based open-source operating system, known for its versatility, customizability, and wide range of distributions. Users with sufficient knowledge can modify and optimize it according to their needs.

95% linux code in written in c and rest in assembly language

**Features of Linux (Short Notes)**

1. **Open Source**: Source code is freely available and modifiable.
2. **Portable**: Runs on various hardware platforms without modification.
3. **Security**: Offers strong user authentication, encryption, and permissions.
4. **Multiuser**: Supports multiple users simultaneously.
5. **Multiprogramming**: Handles multiple programs running at the same time.
6. **Shell**: Provides a command-line interface for user interaction.

### Components of Linux

1. **Hardware**: Physical components like CPU, RAM, storage, and I/O devices.
2. **Kernel**: Core of Linux; manages hardware, processes, memory, device drivers, and file systems.
3. **Shell**: Interface for user commands; examples include Bash, Zsh, and Fish.
4. **System Utilities**: Tools for system management, e.g., file handling (cp, mv), text processing (grep), and system monitoring (top).

**Why Use Linux?**

* **Open Source**: Programmers can create custom operating systems.
* **Secure and Resilient**: Frequent updates and global community support enhance security.
* **Free of Cost**: Available without licensing fees for unlimited installations.
* **Virus-Free**: Resistant to viruses, spyware, and performance slowdowns.
* **Trusted by Corporations**: Used by companies like Google and Facebook for its reliability.

# Cloud computing

Cloud computing is the delivery of IT resources over the Internet on a pay-as-you-go basis, allowing users to access services like computing power, storage, and databases without owning physical servers. It involves storing and accessing data on remote servers instead of local hard drives or servers. Also known as Internet-based computing, cloud computing provides resources as a service through the Internet, enabling users to store various types of data like files, images, and documents.

**SaaS (Software as a Service):**

Definition: Delivers ready-to-use software applications over the internet.

Examples: Gmail, Google Workspace, Dropbox, Salesforce, Zoom.

**PaaS (Platform as a Service):**

Definition: Provides a platform that includes tools and services for developers to build, deploy, and manage applications.

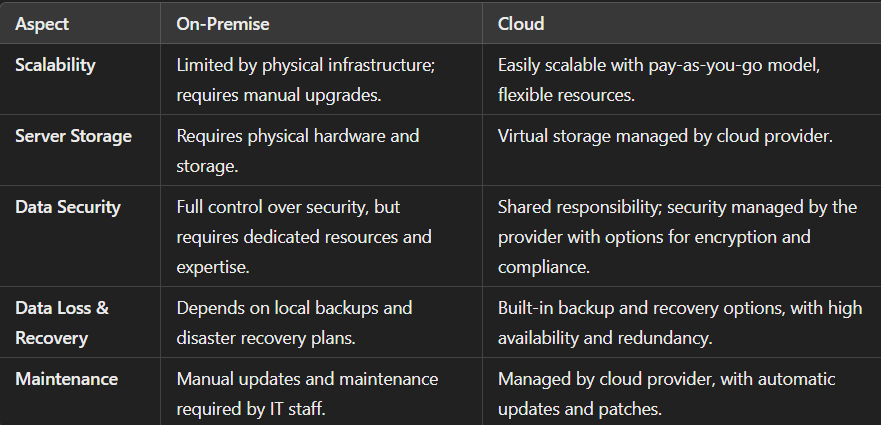
Examples: Firebase, Google App Engine, Microsoft Azure App Service, Heroku, AWS Elastic Beanstalk.

**IaaS (Infrastructure as a Service):**

Definition: Provides virtualized computing resources over the internet, such as servers, storage, and networks.

Examples: Microsoft Azure Virtual Machines, Google Cloud Compute Engine.

Cloud computing services include:



Cloud deployment models:

1. **Private**: Provides enhanced security and customization for specific needs, ideal for companies with strict security and compliance requirements.
2. **Public**: Offers pay-as-you-go scalability and cost-effectiveness, accessible by multiple users.
3. **Hybrid**: Combines private and public clouds, allowing flexible resource optimization between environments.

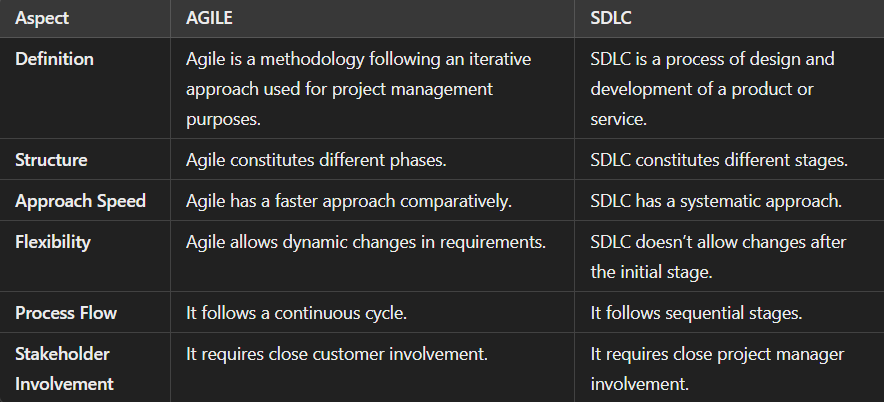
# Software

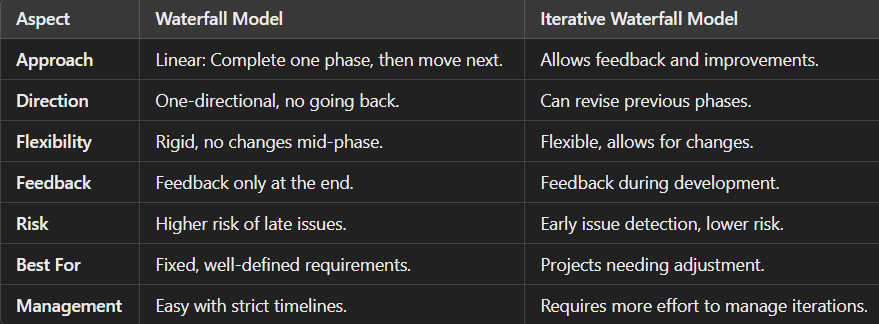
SDLC (Software Development Life Cycle) is a structured process used to develop software systematically. It includes phases like **planning, requirement gathering, designing, development, testing, deployment, and maintenance**, ensuring efficient and high-quality software delivery.

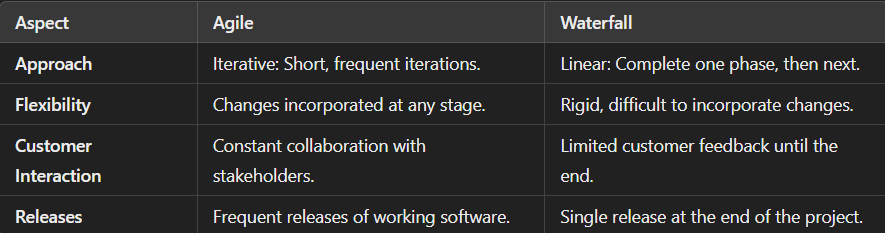
Agile development is a flexible, iterative approach to software development that focuses on **collaboration, customer feedback, and small, frequent releases**. It promotes adaptability to changing requirements and emphasizes teamwork and continuous improvement.

## Life cycle of Agile Methodology

**Plan, design, develop, test, deploy, review, launch.**

****

****

****

# **AI/ML**

### ****What is Artificial Intelligence (AI)?****

Artificial Intelligence (AI) is the simulation of human intelligence in machines that are programmed to think, learn, and make decisions. AI systems aim to perform tasks typically requiring human intelligence, such as reasoning, problem-solving, language understanding, and vision.

### ****What is Deep Learning?****

Deep Learning is a subset of Machine Learning that mimics the structure and function of the human brain using artificial neural networks. It excels in processing large amounts of data and finding patterns that are too complex for traditional algorithms.

* **Structure of ANNs**:
  + **Input Layer**: Accepts the input features.
  + **Hidden Layers**: Perform computations to extract patterns.
  + **Output Layer**: Produces the prediction or classification result.

**Popular Architectures in Deep Learning**

1. **Convolutional Neural Networks (CNNs)**:
   * Used for image recognition and processing.
   * Example: Identifying objects in images.
2. **Recurrent Neural Networks (RNNs)**:
   * Designed for sequential data like time series or text.
   * Example: Language translation, stock price prediction.

**Supervised Learning**

* **Definition**: A type of machine learning where the model is trained on labeled data. The input data comes with corresponding output labels, and the model learns to map inputs to outputs.
* **Examples**:
  + Predicting house prices (regression)
  + Email spam detection (classification)
* **Algorithms**:
  + Linear Regression
  + Logistic Regression
  + Decision Trees
  + Support Vector Machines (SVM)
* **Key Point**: Works with labeled data.

**Unsupervised Learning**

* **Definition**: A type of machine learning where the model learns patterns or structures from unlabeled data. There are no explicit outputs; the goal is to discover hidden patterns.
* **Examples**:
  + Customer segmentation (clustering)
  + Anomaly detection
* **Algorithms**:
  + K-Means Clustering
  + Principal Component Analysis (PCA)
  + Hierarchical Clustering
* **Key Point**: Works with unlabeled data.

**Reinforcement Learning**

* **Definition**: A learning method where an agent interacts with an environment, learns from trial and error, and gets feedback in the form of rewards or penalties.
* **Examples**:
  + Game playing (e.g., AlphaGo)
  + Robotics (e.g., walking robots)
* **Key Components**:
  + **Agent**: Learner or decision-maker.
  + **Environment**: The scenario the agent interacts with.
  + **Actions**: Choices the agent can make.
  + **Reward**: Feedback on the agent's performance.
* **Key Point**: Learns by interacting with the environment to maximize rewards.

A **Confusion Matrix** is a performance measurement tool used in classification problems to evaluate the accuracy of a machine learning model. It shows the number of correct and incorrect predictions made by the model, broken down by class.

**Structure of a Confusion Matrix:**

A typical confusion matrix for a binary classification problem looks like this:

|  | **Predicted Positive** | **Predicted Negative** |
| --- | --- | --- |
| **Actual Positive** | True Positive (TP) | False Negative (FN) |
| **Actual Negative** | False Positive (FP) | True Negative (TN) |

Where:

* **True Positive (TP)**: Correctly predicted positive instances (e.g., model correctly predicted "Yes" when the actual value is "Yes").
* **False Positive (FP)**: Incorrectly predicted positive instances (e.g., model predicted "Yes" when the actual value is "No").
* **False Negative (FN)**: Incorrectly predicted negative instances (e.g., model predicted "No" when the actual value is "Yes").
* **True Negative (TN)**: Correctly predicted negative instances (e.g., model correctly predicted "No" when the actual value is "No").

# Trending technologies

**Artificial Intelligence (AI), Machine Learning (ML) and Deep learning**:

### What is Blockchain Technology?

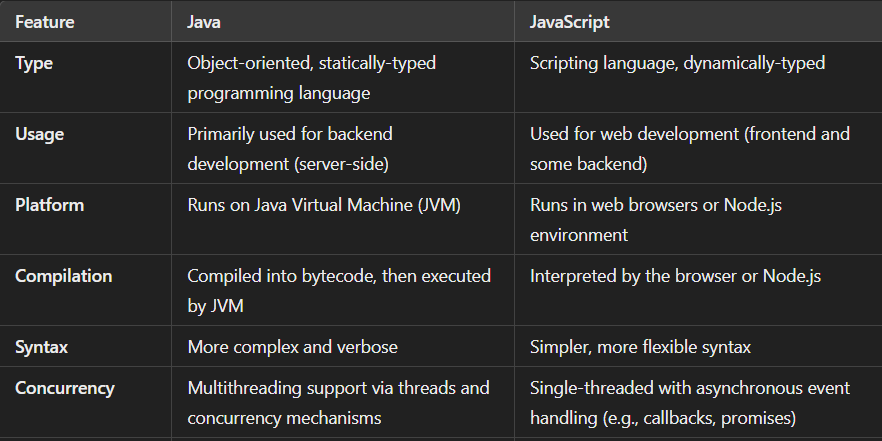
Blockchain is a decentralized, distributed ledger system that securely records transactions across multiple computers in a way that ensures the data cannot be altered retroactively. It operates without the need for a central authority, making it transparent and tamper-proof.

Key Features of Blockchain

1. **Decentralization**: No single entity controls the data; it’s shared across a network.
2. **Immutability**: Once data is recorded, it cannot be changed without consensus.
3. **Transparency**: Transactions are visible to all participants on the network.
4. **Security**: Uses cryptographic methods to secure data and transactions.
5. **Smart Contracts**: Self-executing contracts with predefined rules written into code.

**Internet of Things (IoT)**:

**5G Technology**:



# Projects

**What is an API?**

**API** stands for **Application Programming Interface**. It is a set of rules and protocols that allows one software application to interact with another. APIs enable different systems to communicate with each other by exposing certain functions or data that the system can access.

For example, when you use a weather app, the app calls an API to get weather data from a weather service provider. The API provides an interface to access the data and services without knowing how the backend logic works.

**What is a RESTful API?**

**RESTful API** refers to an API that adheres to the principles of **REST (Representational State Transfer)**, a set of architectural principles for designing networked applications. REST is stateless and is based on standard HTTP methods like GET, POST, PUT, DELETE, etc.

**Key Characteristics of RESTful API:**

1. **Stateless**: Each request from the client to the server must contain all the information needed to understand and process the request. The server does not store any session information between requests.
2. **Client-Server Architecture**: The client and server are separate entities, and they communicate over a network. The client interacts with the API to request resources, while the server handles the processing of those requests.
3. **Uniform Interface**: RESTful APIs provide a consistent and uniform interface, where resources are identified using URLs (Uniform Resource Locators). This makes APIs easier to understand and interact with.
4. **Resources**: In REST, everything is a resource (e.g., user, product, or order), and each resource is accessible via a unique URL. A RESTful API allows you to perform operations on these resources using HTTP methods:
   * **GET**: Retrieve data from the server (e.g., get a list of products).
   * **POST**: Submit data to the server (e.g., create a new product).
   * **PUT**: Update data on the server (e.g., update product details).
   * **DELETE**: Remove data from the server (e.g., delete a product).
5. **Stateless Communication**: Each API call is independent, meaning no session is maintained between the client and server. Each request from the client contains all the necessary information for processing.
6. **Representation**: When you request a resource via a REST API, the server returns a representation of that resource, typically in the form of JSON or XML.

**Example of a RESTful API Request:**

Suppose we have a RESTful API for managing products in an e-commerce store. Here are some examples of RESTful requests:

* **GET** /products: Retrieve a list of all products.
* **GET** /products/123: Retrieve the details of the product with ID 123.
* **POST** /products: Create a new product (send product data in the request body).
* **PUT** /products/123: Update the product with ID 123.
* **DELETE** /products/123: Delete the product with ID 123.

**Difference Between API and RESTful API:**

| **Feature** | **API** | **RESTful API** |
| --- | --- | --- |
| **Definition** | A general interface for software communication | A specific type of API that follows REST principles |
| **Protocol** | Can use any protocol (e.g., SOAP, RPC) | Uses HTTP/HTTPS for communication |
| **Stateless** | May or may not be stateless | Always stateless |
| **Method** | Can use any method or protocol | Uses standard HTTP methods (GET, POST, PUT, DELETE) |
| **Data Format** | Can return any format (e.g., XML, JSON) | Commonly returns data in JSON format |

React is used in projects for its:

1. **Component-based architecture**: breaking them into smaller, independent, reusable units called **components**.
2. **Virtual DOM** for improved performance.

Why mongodb

1. Flexible schema.
2. **Optional Entries:** MongoDB allows each document to have different fields, missing data won't break the structure.

# Watching movie as a hobby

Broaden perceptive, introducing new ideas, relax, explore new culture and ideas