PROGRAM – 3
ExpressJS – Routing, HTTP Methods, Middleware.

a) **AIM: Write a program to define a route, Handling Routes, Route Parameters, Query Parameters and URL building.**

Express.js is a **web application framework for Node.js** that helps you build:

- Web servers
- APIs (Application Programming Interfaces)
- Web apps

Express.js is a minimal and flexible Node.js web application framework that provides a list of features for building web and mobile applications easily. It simplifies the development of server-side applications by offering an easy-to-use API for routing, middleware, and HTTP utilities by providing a robust set of features for handling HTTP requests and responses.

Why Choose Express.js?

Express provides a thin layer of fundamental web application features without effecting Node.js features. It offers:

- A robust routing system
- It simplifies building web servers and APIs.
- Integrates seamlessly with Node.js.
- Offers extensive middleware support to respond to HTTP requests
- A templating engine for dynamic HTML rendering
- Error handling middleware
- Ideal for single-page applications and RESTful APIs.

Definitions & Explanations

Term	Explanation
Route	A route is a URL pattern (like /home, /about) defined in Express to respond to HTTP requests like GET, POST, etc.
Handling Routes	This means writing code that determines what happens when a user visits a specific URL (route).
Route Parameters	Parts of the URL defined using :paramName. Used to pass dynamic values in the URL. Example: /user/:id.
Query Parameters	Key-value pairs sent in the URL after a ?. Example: /search?keyword=nodejs. Used to pass extra data.
URL Building	Dynamically generating a URL by combining base paths, route/query parameters for linking or redirection.

PROGRAM CODE

```
const express = require('express');
const app = express();

// Define a basic route
app.get('/', (req, res) => {
  res.send('Hello! This is the home page.');
});

// Handling a route with a parameter
app.get('/user/:name', (req, res) => {
  const userName = req.params.name;
  res.send(`Hello, ${userName}!`);
});

// Handling a route with query parameters
app.get('/search', (req, res) => {
  const keyword = req.query.q;
  res.send(`You searched for: ${keyword}`);
});

// URL building example
app.get('/url', (req, res) => {
  const fullUrl = req.protocol + '://' + req.get('host') + req.originalUrl;
  res.send(`Full URL: ${fullUrl}`);
});

// Start server
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

COMPILATION & EXECUTION:

To compile and execute the above **Express.js** program, follow these simple steps:

☑ **Step-by-Step Instructions**

1. Install Node.js

- Download and install from: https://nodejs.org
- Verify installation in terminal/command prompt:

```
node -v
npm -v
```

2. Create Your Project Folder
mkdir my-express-app
cd my-express-app

3. Initialize the Project
npm init -y
This creates a package.json file.

4. Install Express
npm install express

5. Create the File
Create a file named 3A.js:
notepad app.js # (Windows)

6. Run the Program
node app.js
☑ You'll see:
Server is running on http://localhost:3000

7. Test It in the Browser

Open your browser and try:

Try These URLs in Browser

URL	What It Does
http://localhost:3000/	Shows "Welcome to the Home Page"
http://localhost:3000/user/AIML	Shows "Hello, AIML"
http://localhost:3000/search?q=express	Shows "You searched for: express"
http://localhost:3000/url	It shows the full URL

b. AIM: Write a program to accept data, retrieve data and delete a specified resource using http methods.

When working with HTTP methods in web applications (like RESTful APIs), we use different methods to interact with resources (data).

Here's how accepting, retrieving, and deleting data works:

1. Accept Data → POST

- Method: POST
- Purpose: Used to send/accept data from the client to the server.
- Example: Submitting a new student record to the database.
- Flow:
  - Client → sends data (e.g., { "name": "Ravi", "rollno": 10 })
  - Server → accepts and stores the data, returns confirmation or the newly created resource.

☑ POST = Create new resource

2. Retrieve Data → GET

- Method: GET
- Purpose: Used to retrieve/fetch data from the server.
- Example: Fetch all student records or get details of a student by ID.
- Flow:
  - Client → sends GET /students
  - Server → responds with data (list of students in JSON, HTML, etc.).

☑ GET = Read resource

3. Delete a Specific Resource → DELETE

- Method: DELETE
- Purpose: Used to remove a specific resource from the server.
- Example: Delete student with ID 10.
- Flow:
  - Client → sends DELETE /students/10
  - Server → removes student with ID 10 and returns success message.

☑ DELETE = Remove resource

What is Postman?
Postman is a popular API (Application Programming Interface) testing tool used by developers, testers, and backend engineers to send, receive, and analyze HTTP requests and responses.

☑ **Why is Postman Used?**

Purpose	Explanation
1. Test APIs easily	You can send GET, POST, PUT, DELETE, etc., requests to any API and see the response instantly.
2. Debug backend issues	Helps find and fix issues in API response, parameters, headers, etc.
3. No code required	You don't need to write a program—just fill in the API URL, method, headers, and body.
4. Save & share requests	You can save your request collections and share with team members.
5. Automate testing	Write pre-request scripts and test cases using JavaScript inside Postman.

 # PROGRAM CODE

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware to parse JSON
app.use(express.json());

// In-memory data storage
let resources = [];

// Route: GET all resources
app.get('/resources', (req, res) => {
  res.json(resources);
});

// Route: POST to add a new resource
app.post('/resources', (req, res) => {
  const resource = req.body;
if (!resource.id || !resource.name) {
    return res.status(400).json({ error: 'Resource must have id and name' });
  }
  // Check for duplicate ID
  const exists = resources.some(r => r.id === resource.id);
  if (exists) {
    return res.status(409).json({ error: 'Resource with this ID already exists' });
  }
  resources.push(resource);
  res.status(201).json({ message: 'Resource added successfully', resource });
});

// Route: DELETE a resource by ID
app.delete('/resources/:id', (req, res) => {
  const id = req.params.id;
```

```
  const index = resources.findIndex(r => r.id === id);

  if (index === -1) {
    return res.status(404).json({ error: 'Resource not found' });
  }

  const deleted = resources.splice(index, 1);
  res.json({ message: 'Resource deleted successfully', deleted });
});

// Start server
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Working with Postman

Using Postman (Graphical Tool)
🔧 Setup
Download & install Postman from https://www.postman.com/downloads

☑ A. POST – Add Data

➕ Add Resource (POST)
URL:http://localhost:3000/resources

Method: POST
Body:
Go to the "Body" tab
Select raw and choose JSON
Enter this JSON:
```
{
  "id": "1",
  "name": "Book"
}
```
Click"Send"
→ Should respond: Item added.

☑ B. GET – Retrieve Data
Method: GET
URL: http://localhost:3000/resources
Click"Send"
→ You'll get the full list of items.
http://localhost:3000/resources---
 {copy this URL in browser & see the result}

☑ C. DELETE – Delete by Name
Delete Resource (DELETE)
URL:http://localhost:3000/resources/1
Method: DELETE
Click"Send"
→ Should respond: Item with name 'pen' deleted.

c. AIM: Write a program to show the working of middleware.

What is Middleware in Express?

Middleware functions are functions that have access to:

- req (request)
- res (response)
- next (function to move to the next middleware)

They run **before** the final route handler and are used for:

- Logging
- Request validation
- Authentication
- Error handling

```
const express = require('express');
const app = express();
const port = 3000;

// -----------------------
// ☑ Middleware Function
// -----------------------
const loggerMiddleware = (req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next(); // Go to next middleware or route handler
};

// -------------------------
// ☑ Apply Middleware
// -------------------------
app.use(loggerMiddleware); // Applies to all routes

// -------------------------
// ☑ Route Handlers
// -------------------------
app.get('/', (req, res) => {
  res.send('Hello, this is the home page!');
});

app.get('/about', (req, res) => {
  res.send('This is the about page.');
});

// -------------------------
// ☑ Start Server
// -------------------------
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

What Happens Here?

- app.use(loggerMiddleware) — applies the logger to all routes.
- Every time you hit a route (like / or /about), the middleware logs the request method and URL before responding.

Test in Browser

Open a browser

- **http://localhost:3000/** → Output: "Hello, this is the home page!"
- **http://localhost:3000/about** → Output: "This is the about page."

PROGRAM – 4
ExpressJS – Templating, Form Data

a. AIM: Write a program using templating engine.

What is a Template Engine?

A **Template Engine** is a tool that helps you **build HTML pages dynamically** using your data. Instead of writing static HTML files manually, a template engine lets you:

- Insert variables (like user names, messages)
- Use logic (like if, for loops)
- Separate your **HTML layout** from your **JavaScript code**

👉 This makes your web app easier to build and maintain.

What is EJS?

EJS stands for **Embedded JavaScript**.

It is a **popular template engine** for Node.js and Express that lets you:

- Embed JavaScript code **inside HTML**
- Generate dynamic web pages on the server

☑ **Features of EJS:**

- Easy to use
- Supports loops and conditionals
- Clean syntax like <%= %> for inserting data

How it Works in Express:

1. You set EJS as the view engine:

   app.set('view engine', 'ejs');

2. Place .ejs files in the /views folder.
3. Use res.render('filename', data) to render the template with dynamic data.

   In web development, `render` means:

   🟤 **To generate and send a final HTML page** to the client (browser), often using a template and dynamic data.

   Example: index.ejs

Step 1: Check Your Views Directory Structure

Make sure you have this folder and file:
C:\Users\ksbam\my-express-app\views\index.ejs

- ☑ views folder **must exist**
- ☑ index.ejs file **must be inside** the views folder

If you're using .ejs, the filename should be: **index.ejs**

```
<!DOCTYPE html>
<html>
<head>
  <title>Home</title>
</head>
<body>
  <h1>Hello from EJS!</h1>
</body>
</html>
```

Step 2: Check That EJS Is Installed
Run this command in your project directory:

npm init -y
npm install ejs

◇ **Step 3: Set the View Engine in 4A.js**
Make sure you have the correct configuration at the top of your 4A.js:

☑ **full code for the program --- 4a.js**

```
const express = require('express');
const path = require('path');
const app = express();

app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

app.get('/', (req, res) => {
  res.render('index');
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

Run the app:
node 4a.js
Open in your browser :
http://localhost:3000

OUTPUT: "Hello from EJS"

B) AIM: Write a program to work with form data.

Step 1: Create a views folder

Make sure the directory C:\Users\ksbam\my-express-app\views exists.
If it doesn't:

1. Open your project folder.
2. Create a folder named views.

Step 2: Create the form view file

Inside the views folder, add a file named:

- If you're using **EJS**: form.ejs

views/form.ejs:

```
<!DOCTYPE html>
<html>
<head>
  <title>Form</title>
</head>
<body>
  <h1>Sample Form</h1>
  <form method="post">
    <input type="text" name="username" placeholder="Enter name" />
    <button type="submit">Submit</button>
  </form>

  <% if (username) { %>
    <h2>You entered: <%= username %></h2>
  <% } %>
</body>
</html>
```

Step 3: Check your app settings

Ensure your Express app sets the views and view engine correctly in 4b.js:

4b.js

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.urlencoded({ extended: true }));
app.set('view engine', 'ejs');

app.get('/', (req, res) => res.render('form', { username: null }));

app.post('/', (req, res) => res.render('form', { username: req.body.username }));

app.listen(3000, () => console.log('Server running at http://localhost:3000'));
```

☑ **Step 4: Install EJS if not installed**
Type the following commands in command prompt

npm init -y
npm install ejs
npm install express ejs body-parser

☑ **Step 5: Run the app**

Start your server:
node 4b.js
Then visit:
👉 **http://localhost:3000**

You should see your form.

OUTPUT:

Sample Form

Enter name Submit

Sample Form

AIML Submit

Sample Form

Enter name Submit

You entered: AIML

PROGRAM – 5
ExpressJS – Cookies, Sessions, Authentication

a. AIM: Write a program for session management using cookies and sessions.

What Is Session Management?
Session management is the process of **tracking user interactions** with a web app across multiple requests.

For example:
When a user logs in, the server remembers them for future visits during that session.

Core Concepts

1. Cookies

- Cookies are small pieces of data stored in the **user's browser**.
- Sent automatically with every request to the server.
- Used to identify users or store small data (like username or session ID).

Example:
res.cookie('username', 'AIML');

2. Sessions

- A session is stored **on the server**.
- The client only stores the **session ID** in a cookie.
- Sessions are **safer** than storing everything in cookies.

Example:
req.session.username = 'AIML;

Authentication Flow Using Cookies & Sessions

1. User logs in

- Client sends username/password to the server.
- Server verifies the credentials.
- If correct:
  - Server creates a session.
  - Session ID is stored in a cookie and sent to the client.

2. User visits protected pages

- On every request, the cookie with session ID is sent.
- Server uses that session ID to retrieve session data.
- If session is valid, user is allowed access.

3. User logs out

- Server deletes the session.
- Cookie is cleared or expired.

How ExpressJS Handles This

Middleware used:

1. **express-session** → handles session creation and tracking
2. **cookie-parser** → parses cookies from incoming requests
3. **body-parser** → parses form data (for login forms)

Benefits of This Approach

- **Security**: Sensitive data is stored on the server.
- **Scalability**: Sessions can be stored in memory, Redis, or a database.
- **Convenience**: Automatically handles login state across multiple pages.

ExpressJS program with:

- ☑ Session creation on login
- ☑ Accessing session data
- ☑ Logout and session destroy

#PROGRAM CODE

☑5A .js

```
const express = require('express');
const session = require('express-session');
const app = express();

app.use(session({
  secret: 'abc',
  resave: false,
  saveUninitialized: true
}));
// Login and set session
app.get('/login', (req, res) => {
  req.session.username = 'Ambika';
  res.send('User logged in');
});

// Access protected page
app.get('/profile', (req, res) => {
  if (req.session.username) {
    res.send(`Welcome ${req.session.username}`);
  } else {
    res.send('Please login first');
  }
});
```

// Logout and destroy session

```
app.get('/logout', (req, res) => {
  req.session.destroy();
  res.send('User logged out');
});
app.listen(3000, () => {
  console.log('http://localhost:3000');
});
```

🔗 How to Use Execute :

Steps:

1. **node -v**
2. **npm -v**

Setup Instructions

First, install dependencies:

1. **npm list express**
2. **npm install express express-session**
3. **npm init -y**

Check if express-session is installed locally:

npm list express-session

1. Visit http://localhost:3000/**login → Session created**
2. Visit http://localhost:3000/**profile → Shows username**
3. Visit http://localhost:3000/**logout → Session destroyed**
4. Visit /profile again → Asks to login again

b. AIM: Write a program for user authentication.

ExpressJS Authentication Flow (using Sessions and Cookies)

☑ **Step-by-Step Flow:**

1. User opens login page
2. User enters username and password
3. Server checks credentials
4. **If correct:**
   - Server creates a **session**
   - Stores user info in session (on the server)
   - Sends a **cookie** with a session ID to the browser
5. **For future requests:**
   - Cookie is sent automatically
   - Server checks session ID
   - If valid, user is considered logged in

❁ **Key Tools in ExpressJS:**

Tool	Purpose
express-session	Creates and manages sessions
cookie-parser	Parses cookies from HTTP requests
body-parser	Parses form data (POST requests)

 # PROGRAM CODE

```
const express = require('express');
const bodyParser = require('body-parser');
const session = require('express-session');

const app = express();
const PORT = 3000;

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(session({
```

```
  secret: 'secret-key',
  resave: false,
  saveUninitialized: true
}));

// Dummy user credentials
const USER = {
  username: 'admin',
  password: '1234'
};

// Login page
app.get('/', (req, res) => {
  res.send(`
    <h2>Login</h2>
    <form method="POST" action="/login">
      <input type="text" name="username" placeholder="Username" required/><br/>
      <input type="password" name="password" placeholder="Password" required/><br/>
      <button type="submit">Login</button>
    </form>
  `);
});

// Login handler
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  if (username === USER.username && password === USER.password) {
    req.session.user = username;
    res.send(`<h2>Welcome, ${username}!</h2><a href="/logout">Logout</a>`);
  } else {
    res.send('<h3>Invalid credentials. <a href="/">Try again</a></h3>');
  }
});
```

```
// Logout handler
app.get('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/');
});

// Start server
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Execution Steps:

1. **node -v**
2. **npm -v**

Setup Instructions

First, install dependencies:

3. **npm list express** (if version number is shown as empty u have to install express) i.e, --→ npm install express
4. **npm install express ejs body-parser**
5. **npm install express express-session**
6. **npm init -y**

Test with Browser:

1. Visit http://localhost:3000 to see the login form.
2. Enter:
   - **Username**: admin
   - **Password**: 1234
3. You'll be logged in with a welcome message.

Note: If wrong user name & password is entered it shows invalid credentials.

PROGRAM-6

ExpressJS – Database, RESTful APIs

a .Write a program to connect MongoDB database using Mongoose and perform CRUD operations.

What is MongoDB?

MongoDB is a **NoSQL database** that stores data in a **document-oriented format** (JSON-like documents) instead of traditional **rows and tables** like SQL databases.
It's widely used in **modern web and mobile apps** because it is **fast, scalable, and flexible**

What is JSON?

The full form of JSON is:

👉 **JavaScript Object Notation**

- A **lightweight data-interchange format**.
- Used to **store and exchange data** between a server and a client.
- Easy to read and write for humans.
- Easy to parse and generate for machines.

◇ **Example of JSON**

```
{
 "name": "Alice",
 "age": 25,
 "email": "alice@example.com",
 "isStudent": false
}
```

Key Features of MongoDB

1. **Document-oriented**
2. **Collections instead of Tables**
   - Documents are grouped into **collections** (like tables in SQL).
   - Example: A users collection can have many documents.
3. **Schema-less (Flexible)**
   - Unlike SQL, you don't have to predefine a strict schema.
   - One user document may have { name, email, age }, another may have { name, phone }.

Mongoose is a popular **Object Data Modeling (ODM) library** for MongoDB and Node.js.

What Mongoose Does

- Acts as a **bridge** between your Express app (Node.js) and **MongoDB database**.
- Provides a **schema-based solution** to model your data (like defining structure, validation rules, and relationships).
- Makes working with MongoDB easier using **JavaScript objects**, instead of writing raw queries.

◇ **Why Use Mongoose with Express?**

1. **Schema definition** – You can define how your MongoDB documents should look.
2. **Validation** – Ensures data integrity before saving into MongoDB.
3. **Query helpers** – Easier to query, update, and delete data.
4. **Middleware (hooks)** – Functions that run before/after saving, updating, or deleting.
5. **Relationships & population** – Manage references between collections.

CRUD operations

CRUD operations are the **fundamental building blocks** of working with any database or backend system.

CRUD stands for the four basic operations you can perform on data in a database:

◇ **CRUD = Create, Read, Update, Delete**

- **Create** → Add a new contact.
- **Read** → View your contacts.
- **Update** → Edit a contact's phone number.
- **Delete** → Remove a contact.

Install MongoDB locally

download from: https://www.mongodb.com/try/download/community

Verify installation : **mongod --version**

After installing:

- Start MongoDB server:

This starts MongoDB on mongodb://127.0.0.1:27017 (default port).

Steps: Install Required Packages

Run in terminal:

1. node -v
2. npm -v
3. npm install express mongoose
4. npm list mongoose
5. **mongod –version**
6. npm init -y

 # PROGRAM CODE

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
app.use(express.json()); // for JSON data

// 1 Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/mydb')
  .then(() => console.log('✅ MongoDB Connected'))
  .catch(err => console.log(err));

// 2 Create Schema & Model
const UserSchema = new mongoose.Schema({
  name: String,
  email: String
});
const User = mongoose.model('User', UserSchema);

// 3 CRUD Routes
// CREATE (POST)
app.post('/users', async (req, res) => {
  const user = new User(req.body);
  await user.save();
  res.send(user);
});
// READ (GET)
app.get('/users', async (req, res) => {
  const users = await User.find();
  res.send(users);
});

// UPDATE (PUT)
app.put('/users/:id', async (req, res) => {
  const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
  res.send(user);
```

```
});
// DELETE (DELETE)
app.delete('/users/:id', async (req, res) => {
  await User.findByIdAndDelete(req.params.id);
  res.send({ message: 'User deleted' });
});

// 4 Start Server
app.listen(3000, () => console.log('🚀 Server running on port 3000'));
```

Step : Run the Project

node 6a.js

☑ You can test the API using **Postman**

Explanation:

Step 1: Create a User (POST)
Request: POST : http://localhost:3000/users

Body (JSON):

```
{
  "name": "Alice",
  "email": "alice@test.com"
}
```

Response Example:

```
{
  "_id": "66a90f4a8c1b3f18f92c1a23",
  "name": "Alice",
  "email": "alice@test.com",
  "__v": 0
}
```

Step 2: Update the User (PUT)
Use the _id from the response above in your **PUT request**:
PUT http://localhost:3000/users/66a90f4a8c1b3f18f92c1a23
Body (JSON):

```
{
  "name": "Alice Updated",
  "email": "alice.new@test.com"
}
```

Step 3: Check Update
Then you can GET all users to confirm:
GET http://localhost:3000/users

Step 4: For Delete

DELETE http://localhost:3000/users/66a90f4a8c1b3f18f92c1a23

b. AIM: Write a program to develop a single page application using RESTful APIs.

What is a RESTful API?

A **RESTful API** (Representational State Transfer API) is a way for two systems (like a client and server) to communicate over the internet using **HTTP methods**.

Example: RESTful API for Users

HTTP Method	Endpoint	Description
GET	/users	Get all users
GET	/users/1	Get user with ID = 1
POST	/users	Create a new user
PUT/PATCH	/users/1	Update user with ID = 1
DELETE	/users/1	Delete user with ID = 1

1. Install dependencies

- **npm init -y**
- **npm install express body-parser**

2. 6b.js

```
const express = require('express');
const app = express();
app.use(express.json());
app.use(express.static('public'));

let items = ["Apple", "Banana"];

app.get('/api/items', (req, res) => res.json(items));
app.post('/api/items', (req, res) => { items.push(req.body.name); res.json(items); });

app.listen(3000, () => console.log('http://localhost:3000'));
```

3. public/index.html (Single Page App)

Create a folder named **public** in your own created folder and save this file as index.html in public folder

```
<!DOCTYPE html>
<html>
<body>
  <h2>Items</h2>
```

```
  <ul id="list"></ul>
  <input id="item" placeholder="New item">
  <button onclick="addItem()">Add</button>

  <script>
    async function load(){
      const res = await fetch('/api/items');
      document.getElementById('list').innerHTML = (await res.json()).map(i=>`<li>${i}</li>`).join('');
    }
    async function addItem(){
        await      fetch('/api/items',{method:'POST',headers:{'Content-
Type':'application/json'},body:JSON.stringify({name:document.getElementById('item').value})});
      document.getElementById('item').value='';
      load();
    }
    load();
  </script>
</body>
</html>
```

To Run:

node 6b.js

Output:

PROGRAM-7

ReactJS – Render HTML, JSX, Components – function & Class

What is React?

- React is a JavaScript library for building user interfaces.
- React is used to build single-page applications.
- React is a tool for building UI components.
- React allows us to create reusable UI components.
- React is a front-end JavaScript library.
- React was developed by the Facebook Software Engineer Jordan Walke.
- React is also known as React.js or ReactJS.

How does React Work?

- React creates a VIRTUAL DOM in memory.
- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.
- React only changes what needs to be changed!
- React finds out what changes have been made, and changes **only** what needs to be changed.

Advantages of React

- ✓ Faster UI updates with Virtual DOM
- ✓ Reusable and maintainable code through components.
- ✓ Large community and ecosystem (React Router, Redux, Next.js, etc.).
- ✓ Easy to learn if you know JavaScript.
- ✓ Works for both **web** (React) and **mobile apps** (React Native).

React vs Framework

- React is a library (not a full framework).
- It only handles the view layer (UI).
- For routing, state management, and APIs, we use additional tools.

Real-World Usage

- Used by big companies → Facebook, Instagram, Netflix, Uber, Airbnb, etc.
- Suitable for:
  - Dashboards
  - Social media apps
  - E-commerce websites
  - Real-time applications (chat, video apps)

a) AIM: Write a program to render HTML to a web page.

React Render HTML

- React's goal is in many ways to render HTML in a web page.
- React renders HTML to the web page via a container, and a function called **createRoot().**

The createRoot Function

- The createRoot function is located in the main.jsx file in the src folder, and is a built-in function that is used to create a root node for a React application
- The **createRoot()** function takes one argument, an HTML element.
- The purpose of the function is to define the HTML element where a React component should be displayed.

The Container

- React uses a container to render HTML in a web page.
- Typically, this container is a element in the index.html file.

The render Method

- The render method defines what to render in the HTML container.
- The result is displayed in the element.

Setting up a React Environment

- First, make sure you have Node.js installed. You can check by running this in your terminal:
  node -v
- If Node.js is installed, you will get a result with the version number:
  v22.15.0
- If not, you will need to install Node.js.

Build Tool (Vite)

Vite is a modern frontend build tool that helps developers set up and run web projects very quickly.

The word **"Vite"** comes from **French**, meaning **"fast"**.
Its main focus is **speed**: faster startup and faster builds compared to older tools.

Vite = A **fast tool** to build and run modern web projects.

- **Dev server**: Instant updates when coding.
- **Build tool**: Optimizes code for production.
- **Supports** React, Vue, Svelte, etc.
- Name "Vite" = French for **fast**.

Install a Build Tool (Vite)

- When you have Node.js installed, you can start creating a React application by choosing a build tool.
- Run this command to install Vite:

  npm install -g create-vite

- If the installation was a success, you will get a result like this:

  added 1 package in 649ms

Create a React Application

Run this command to create a React application named my-react-app:

npm create vite@latest my-react-app -- --template react

If you get this message, just press y and press Enter to continue:

Need to install the following packages

create-vite@6.5.0
Ok to proceed? (y)

If the creation was a success, you will get a result like this:

> npx

> create-vite my-react-app --template react

o Scaffolding project in C:\Users\stale\my-react-app...

— Done.

Install Dependencies

- As the result above suggests, navigate to your new react application directory:

cd my-react-app

- And run this command to install dependencies:

npm install

Which will result in this:

added 154 packages, and audited 155 packages in 8s

33 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

Run the React Application

- Now you are ready to run your first *real* React application!
- Run this command to run the React application my-react-app:

npm run dev

Which will result in this:

VITE v6.3.5 ready in 217 ms

➜ Local: http://localhost:5173/
➜ Network: use --host to expose
➜ press h + enter to show help

Rendering:

In **React**, **rendering** means **taking React elements (JSX or components) and displaying them inside the browser's DOM**.

Step-by-Step Explanation of the Example

1. HTML Setup

```
<div id="root"></div>
```

- This empty is where React will show all the content.
- Think of it as the **container** for React.

2. Import React and ReactDOM

- React → helps us create components using JSX.
- ReactDOM → is responsible for actually rendering (showing) those components inside the browser.

3. Create a Component

```
function Hello() {
  return <h1>Hello, React Rendering!</h1>;
}
```

- This is a **function component**.
- It returns **JSX** (<h1>...</h1>), which looks like HTML but is actually JavaScript.

4. Render to DOM

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Hello />);

- **document.getElementById("root")** → finds the .
- **ReactDOM.createRoot(...)** → tells React where to put the UI.
- **.render(<Hello />)** → renders the Hello component inside that div.

✅ So what happens?

- React takes <Hello /> → replaces it with <h1>Hello, React Rendering!</h1>
- Then inserts it inside
- Finally, the user sees **Hello, React Rendering!** on the webpage.

👉 In short:

1. **Component** creates UI (JSX).
2. **ReactDOM** puts that UI inside the HTML page.

Look in the my-react-app directory, and you will find a src folder. Inside the src folder there is a file called **main.js**, open it and it will look like this:

Example:

The default content of the src/main.jsx file is replaced with the following content

main.jsx

```
import { createRoot } from 'react-dom/client'

createRoot(document.getElementById('root')).render(
  <h1>Hello React!</h1>
)
```

Output: Hello React!

Example

Display a paragraph inside the "root" element:

main.jsx

```
import { createRoot } from 'react-dom/client'

createRoot(document.getElementById('root')).render(
  <p>Welcome!</p>
)
```

Output:
Welcome

b. AIM: Write a program for writing markup with JSX

What is JSX?

- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.

Coding JSX

- JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
- JSX converts HTML tags into react elements.
- You are not required to use JSX, but JSX makes it easier to write React applications.

main.jsx

```
const myElement = <h1>I Love JSX!</h1>;

createRoot(document.getElementById('root')).render(
  myElement
);
```

JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

Writing markup with JSX

JSX is a syntax extension for JavaScript, used with React to describe what the UI should look like. This guide will explain why React mixes markup with rendering logic, how JSX differs from HTML, and how to display information using JSX.

How JSX is Different from HTML?

JSX may look similar to HTML, but there are key differences:

- **JavaScript Expressions:** JSX allows you to embed JavaScript expressions inside curly braces {}. This lets you dynamically render content based on your application state or props.
- **Attributes**: JSX attributes follow camelCase naming conventions instead of the kebab-case used in HTML. For example, class in HTML becomes className in JSX.
- **Self-Closing Tags**:Like XML, JSX requires that tags be properly closed. If an element doesn't have children, you must use a self-closing tag (e.g., <input /> instead of <input>).
- **Fragment Syntax**: JSX supports fragment syntax, allowing you to group multiple elements without adding extra nodes to the DOM. This can be done using <React.Fragment> or shorthand syntax <> </>.

Rules of JSX

1. Return a single root element

To return multiple elements from a component, enclose them within a single parent element.

```
<div>
  <h1>Geeks for Geeks</h1>
  <ul>
    ......
  </ul>
</div>
```

2. Close all the Tags

JSX mandates explicit closure of tags: self-closing tags such as `<img>` must be written as `<img />`, and wrapping tags like `<li>oranges` should be corrected to `<li>oranges</li>`.

```
<>
<img   src= "https://google.com/photos.jpg"
         alt =" google "
         class = "photo"
/>
   <ul>
        <li> Blue </li>
        <li> Green </li>
        <li>  Red </li>
  </ul>
</>
```

3. camelCase most of the things

JSX converts into JavaScript, where JSX attributes become keys in JavaScript objects. To handle this conversion seamlessly, React adopts camelCase for attribute names. For instance, attributes like stroke-width in JSX are transformed into strokeWidth in JavaScript. Similarly, to avoid conflicts with reserved words like class, React uses className instead, aligning with corresponding DOM properties for clarity and compatibility.

```
<img
       src= "https://imgur.com/"
       alt= "gfg"
       className = "photo"
/>
```

Step -by- step procedure:

Look in the my-react-app directory, and you will find a src folder. Inside the src folder there is a file called **main.js**, open it and replace with your required content.

❖ **Example 1**

Main.JSX:

```
const myElement = <h1>I Love JSX!</h1>;

createRoot(document.getElementById('root')).render(
  myElement
);
```

Output: I Love JSX!

Example 2: Expressions in JSX

With JSX you can write expressions inside curly braces { }.

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

Execute the expression 5 + 5:

main.jsx

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```

Output: React is 10 times better with JSX

Example 3: Inserting a Large Block of HTML

To write HTML on multiple lines, put the HTML inside parentheses:

Create a list with three list items:

main.jsx

```
const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

Output:

- Apples
- Bananas
- Cherries

Example 3: One Top Level Element

The HTML code must be wrapped in *ONE* top level element.

So if you like to write *two* paragraphs, you must put them inside a parent element, like a div element.

Example

Wrap two paragraphs inside one DIV element

main.jsx

```
const myElement = (
  <div>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </div>
);
```

Output:

I am a paragraph.
I am a paragraph too.

- JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.
- Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.
- A fragment looks like an empty HTML tag: <></>.

Example
Wrap two paragraphs inside a fragment:

main.jsx

```
const myElement = (
  <>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </>
);
```

Output:

I am a paragraph.
I am a paragraph too.

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Example

Close empty elements with />

main.jsx

```
const myElement = <input type="text" />;
```

Output:

Note: JSX will throw an error if the HTML is not properly closed.

Attribute class = className

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute **className** instead.

JSX solved this by using className instead. When JSX is rendered, it translates className attributes into class attributes.

Example

Use attribute className instead of class in JSX:

main.jsx

```
const myElement = <h1 className="myclass">Hello World</h1>;
```

Comments in JSX

Comments in JSX are written with {/* */}

Example

main.jsx

```
const myElement = <h1>Hello {/* Wonderful */} World </h1>;
```

Output:

Hello World

c. AIM: Write a program for creating and nesting components (function and class).

React Components

What is a Component?

- A **component** is a reusable, independent piece of UI in React.
- Each component can represent part of a webpage (like a button, header, or form).
- Components make code **modular, reusable, and easier to maintain**.

Types of Components

React provides two main ways to create components:

a) Function Components

- A function that returns JSX.
- Simple and lightweight.
- Can accept **props** (input data) and display them.
- In modern React, hooks (useState, useEffect, etc.) make function components powerful.

Example:

```
function Greeting(props) {
  return <h2>Hello, {props.name}!</h2>;
}
```

b) Class Components

- Written as JavaScript **classes**.
- Must extend React.Component.
- Have a **render()** method that returns JSX.
- Can manage their own **state** and lifecycle methods.

Example:

```
class Welcome extends React.Component {
  render() {
    return <h1>Welcome, {this.props.name}!</h1>;
  }
}
```

3. Nesting Components

- Nesting means **using one component inside another**.
- This helps build complex UIs by combining small, reusable components.

Example:

```
function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Welcome name="Bob" />
    </div>
  );
}
```

Here:

- App is the parent component.
- Greeting and Welcome are **nested (child) components** inside App.

4. Props and State in Nesting

- **Props** (properties) → Passed from parent to child. They are read-only.
- **State** → Local to a component; used to manage changing data.
- Nested components usually **receive data via props** from parent components.

5. Why Use Nesting?

✅ Reusability → One component can be used in many places.
✅ Separation of Concerns → Each component handles its own logic/UI.
✅ Easy Maintenance → Small, independent units are easier to debug.
✅ Clear Hierarchy → Parent → Child → Sub-child structure makes UI structured.

📌 Summary:

- Function components → simple, use hooks.
- Class components → use render() and lifecycle methods.
- Nesting → placing one component inside another to build a hierarchy.

📌 Example: Creating & Nesting Function and Class Components

```
// Import React and ReactDOM
import React from "react";
import ReactDOM from "react-dom/client";
```

```
// ✅ Function Component

function Greeting(props) {
  return <h2>Hello, {props.name}!</h2>;
}

// ✅ Class Component

class Welcome extends React.Component {
  render() {
    return (
      <div>
        <h1>Welcome Component</h1>
        {/* Nesting function component inside class */}
        <Greeting name="Alice" />
        <Greeting name="Bob" />
      </div>
    );
  }
}

// ✅ Another Function Component nesting both
function App() {
  return (
    <div>
      <h1>App Component</h1>
      {/* Nesting class component */}
      <Welcome />
    </div>
  );
}

// Render into the root element
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

Output:

```
App Component
Welcome Component
Hello, Alice!
Hello, Bob!
```

🔍 **Explanation:**

1. **Greeting** → A simple **function component** with props.
2. **Welcome** → A **class component** that **nests Greeting** multiple times.
3. **App** → The main function component, which **nests Welcome**.
4. **root.render(<App />)** → Renders everything into the HTML page.

PROGRAM-8

ReactJS – Props and States, Styles, Respond to Events

a. AIM: Write a program to work with props and states.

In React, State allows components to manage and update internal data dynamically, while Props enables data to be passed from a parent component to a child component. Understanding their differences and use cases is essential for developing efficient React applications.

State in React

- React components has a built-in state object.
- The state object is where you store property values that belong to the component.
- When the state object changes, the component re-renders.

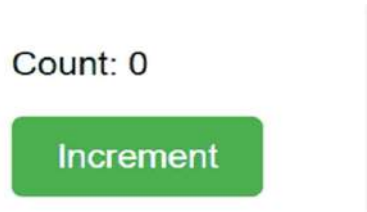
State is a built-in object in React components that holds data or information about the component. It is mutable, which means it can be updated within the component using the setState method in class components or the useState hook in functional components.

- State is local to the component and cannot be accessed by child components unless passed down as props.
- It is mutable, meaning it can change over time based on user interactions or API responses.
- When state updates, the component re-renders to reflect the changes.
- Managed using useState in functional components or this.setState in class components.

 # PROGRAM

```
import React, { useState } from 'react';

function Counter() {
   const [count, setCount] = useState(0);

   return (
      <div>
         <p>Count: {count}</p>
         <button onClick={() => setCount(count + 1)}>Increment</button>
      </div>
   );
}

export default Counter;
```

Output



In this example

- useState(0) initializes the state variable count with the value 0.
- The setCount function is used to update the state whenever the button is clicked. This triggers a re-render, updating the displayed count.

Props in React

- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.
- props stand for properties.
- React Props are like function arguments in JavaScript *and* attributes in HTML.

Props (short for Properties) are used to pass data from a parent component to a child component. Unlike state, props are immutable, meaning they cannot be modified within the receiving component.

- Props allow components to be reusable and dynamic.
- Props are read-only and cannot be changed by the child component.
- They help in data communication between components.
- Passed as attributes in JSX elements.
- import React from 'react';

 # PROGRAM CODE

```
function Greeting({ name }) {
   return <h1>Hello, {name}!</h1>;
}

function App() {
   return <Greeting name="Jiya" />;
}

export default App;
```

Output

Hello, Jiya!

Props in React

Here, the Greeting component receives a prop named name and displays a personalized message. The App component passes the value "Jiya" as a prop.

b. AIM: Write a program to add styles (CSS & Sass Styling) and display data

CSS in React is used to style components, just like in normal HTML, but React lets you organize styles in multiple ways:

1. **Normal CSS** – Create a .css file and import it into a component.
2. **Inline Styling** – Add styles directly as a JavaScript object using the style attribute.
3. **CSS Modules** – Scoped CSS files (.module.css) that avoid class name conflicts.
4. **Sass (SCSS)** – Advanced styling with variables, nesting, and mixins using .scss files.

CSS makes React components look good, and React gives you flexible ways to apply it.

What is Sass?

- Sass stands for Syntactically Awesome Stylesheet
- Sass is an extension to CSS
- Sass is a CSS pre-processor
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
- Sass is free to download and use

Why Use Sass?

Stylesheets are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help.

Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.

Sass File Type

Sass files has the **".scss"** file extension.

Sass Comments

Sass supports standard CSS comments /* comment */, and in addition it supports inline comments // comment:

Sass Example

```
/* define primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;

/* use the variables */
.main-header {
  background-color: $primary_1; // here you can put an inline comment
}
```

Step -by -step Procedure:

1. Folder Setup

```
my-app/
 ├ src/
 │  ├ App.js
 │  ├ App.css
 │  ├ App.scss
 │  └ index.js
```

Step 2: Create a React Project

npx create-react-app my-app

- my-app is your project folder.
- It will take a few minutes to set up.

Step 3: Navigate to Your Project

cd my-app

Step 4: Install Sass

npm install sass

- This allows React to compile .scss files automatically.

Step 5: Create Your Files

5a. App.css : (Inside src folder)

```
container {
  text-align: center;
  background-color: #f9f9f9;
  padding: 20px;
```

```
}
title {
  color: #2c3e50;
  font-size: 2rem;
}
```

5b. App.scss : (Inside src folder)

```
$list-color: #27ae60;

.data-list {
  list-style: none;
  padding: 0;

  li {
    background: lighten($list-color, 30%);
    margin: 8px 0;
    padding: 10px;
    border-radius: 6px;
    font-weight: bold;

    &:hover {
      background: darken($list-color, 10%);
      color: #fff;
      cursor: pointer;
    }
  }
}
```

5c. App.js (Inside src folder)

```
import React from "react";
import "./App.css";   // CSS
import "./App.scss";  // Sass

function App() {
  const fruits = ["Apple", "Banana", "Cherry", "Mango", "Orange"];

  return (
    <div className="container">
      <h1 className="title">Fruit List</h1>
      <ul className="data-list">
        {fruits.map((fruit, index) => (
          <li key={index}>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

Explanation

1. App.js imports CSS and SCSS for styling.
2. The fruits array is mapped to a list of <li> elements.
3. App.css contains basic styling like text alignment and title color.
4. App.scss uses Sass features like variables ($fruit-color) and nested styles for .fruit-list .fruit-item.
5. When you hover over a fruit, the color changes thanks to the &:hover in Sass.

Execution steps:

Open the terminal and follow the below commands

1. **npx create-vite my-react-app --template react**
2. **cd my-react-app**
3. **npm install**
4. **npm install sass**
5. **npm run dev**

-→

VITE v7.1.6 ready in 508 ms

➜ Local: http://localhost:5173/
➜ Network: use --host to expose
➜ press h + enter to show help

Copy the url: **http://localhost:5173/** in any web browser and it displays the below result.

Output:

```
--------------------------------------
|                                    |
|          Fruit List                |
|                                    |
|   [Apple]                          |
|   [Banana]                         |
|   [Cherry]                         |
|   [Mango]                          |
|   [Orange]                         |
|                                    |
--------------------------------------
```

c. AIM: Write a program for responding to events.

In React, events are actions that occur within an application, such as clicking a button, typing in a text field, or moving the mouse. React provides an efficient way to handle these actions using its event system.

React event handlers are written inside curly braces:

Syntax

<button onClick={shoot}>Take the Shot!</button>

Event handlers like onClick, onChange, and onSubmit are used to capture and respond to these events.

Syntax

<element onEvent={handlerFunction} />

- element: The JSX element where the event is triggered (e.g., <button>, <input>, etc.).
- onEvent: The event name in camelCase (e.g., onClick, onChange).
- handlerFunction: The function that handles the event when it occurs.

Commonly Used React Event Handlers

React provides a variety of built-in event handlers that we can use to handle different user interactions:

React Event	Description
onClick	This event is used to detect mouse clicks in the user interface.
onChange	This event is used to detect a change in the input field in the user interface.
onSubmit	This event fires on the submission of a form in the user interface and is also used to prevent the default behavior of the form.
onKeyDown	This event occurs when the user press any key from the keyboard.
onKeyUp	This event occurs when the user releases any key from the keyboard.
onMouseEnter	This event occurs when the ouse enters the boundary of the element

PROGRAM CODE

// App.jsx

```
import React, { useState } from "react";

function App() {
  // State to store input value
  const [name, setName] = useState("");
  // State to store button click message
  const [message, setMessage] = useState("");

  // Event handler for button click
  const handleClick = () => {
    setMessage(`Hello, ${name || "Guest"}!`);
  };

  // Event handler for input change
  const handleChange = (event) => {
    setName(event.target.value);
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>React Event Handling Example</h1>

      <input
        type="text"
        placeholder="Enter your name"
        value={name}
        onChange={handleChange}
        style={{ padding: "10px", fontSize: "16px" }}
      />
      <br /><br />

      <button
        onClick={handleClick}
        style={{ padding: "10px 20px", fontSize: "16px", cursor: "pointer" }}
      >
        Greet Me
      </button>
      <br /><br />

      <h2>{message}</h2>
    </div>
  );
}

export default App;
```

Output:

- Type your name in the input box → click **Greet Me** → the message "Hello, [Your Name]!" appears

PROGRAM-9
ReactJS – Conditional Rendering, Rendering Lists, React Forms

a. AIM: Write a program for conditional rendering.

What is Conditional Rendering?

Conditional rendering in React allows you to render different components or elements based on a condition. It's like using an if-else statement in HTML—React decides what to display depending on the state or props.

Components of the Program

State Management (useState):

```
const [isLoggedIn, setIsLoggedIn] = useState(false);
```

- Here, **isLoggedIn** is a state variable that tracks whether the user is logged in (true) or not (false).
- **setIsLoggedIn** is a function to update the state.

Conditional Rendering with Ternary Operator:

```
{isLoggedIn ? (
  <p>Welcome, User! You are logged in.</p>
) : (
  <p>Please log in to continue.</p>
)}
```

- If isLoggedIn is true, it displays a welcome message.
- If isLoggedIn is false, it asks the user to log in.
- This is the core part of conditional rendering.

Event Handling (onClick):

```
<button onClick={handleLoginToggle}>
  {isLoggedIn ? "Logout" : "Login"}
</button>
```

- Clicking the button toggles the isLoggedIn state.
- The button text also changes based on the state.

Dynamic UI Update:
React automatically re-renders the component whenever the state changes. So, when the user logs in or out, the displayed message and button text update immediately.

Why It's Useful
- Conditional rendering allows dynamic user interfaces.
- You can show or hide elements, switch components, or change content depending on user actions, API responses, or application state.
- It keeps the UI interactive and responsive without reloading the page.

PROGRAM CODE:

A **simple ReactJS example** for **conditional rendering** using a functional component.

```
import React, { useState } from "react";

function App() {
  // State to track whether the user is logged in
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  // Function to toggle login status
  const handleLoginToggle = () => {
    setIsLoggedIn(!isLoggedIn);
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Conditional Rendering Example</h1>

      {/* Conditional rendering using ternary operator */}
      {isLoggedIn ? (
        <p>Welcome, User! You are logged in.</p>
      ) : (
        <p>Please log in to continue.</p>
      )}

      {/* Button to toggle login status */}
      <button onClick={handleLoginToggle} style={{ padding: "10px 20px", marginTop: "20px" }}>
        {isLoggedIn ? "Logout" : "Login"}
      </button>
    </div>
  );
}

export default App;
```

Output Example

Initially:

Please log in to continue.

[Login]

After clicking Login:

Welcome, User! You are logged in.

[Logout]

b) AIM: Write a program for rendering lists.

What is List Rendering?

In React, list rendering is the process of displaying multiple items from an array or collection dynamically in the UI. Instead of manually writing each item, React allows you to loop through the data and render it efficiently.

Components of the Program

Array of Items:

```
const fruits = ["Apple", "Banana", "Cherry", "Mango", "Orange"];
```

- This is the data source.
- Each element in the array will be rendered as a list item in the UI.

Using map() to Render Lists:

```
{fruits.map((fruit, index) => (
  <li key={index}>{fruit}</li>
))}
```

- The map() function iterates over the array.
- For each element, it returns a React element (<li> in this case).
- key={index} is a unique identifier required by React to optimize rendering and efficiently update the UI when items change.

Dynamic UI Rendering:

When the array changes (items added, removed, or updated), React automatically updates the displayed list. This keeps the interface reactive and up-to-date.

Using <ul> for List Structure:

```
<ul> ... </ul>
```

The <ul> (unordered list) element is used to display the list items visually in a bullet format**.**

Why It's Useful

- Dynamically renders any number of items from arrays or objects.
- Efficiently updates the DOM when data changes without reloading the page.
- Essential for building dynamic UIs like menus, tables, and lists from APIs or databases.

PROGRAM CODE

A simple ReactJS example for rendering lists using a functional component.

```
import React from "react";

function App() {
  // Array of fruits
  const fruits = ["Apple", "Banana", "Cherry", "Mango", "Orange"];

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Rendering Lists Example</h1>

      <ul>
        {/* Rendering list items using map() */}
        {fruits.map((fruit, index) => (
          <li key={index}>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Output

The program will render:

- Apple
- Banana
- Cherry
- Mango
- Orange

Each element of the array is displayed as a **list item** in the browser.

c) AIM: Write a program for working with different form fields using react forms.

What are React Forms?

In React, forms are used to **collect user input**. React forms can be either **controlled** or **uncontrolled**. Controlled forms are recommended because the form data is **stored in React state**, making it easier to manage, validate, and update dynamically.

Components of the Program

State Management (useState):

```
const [formData, setFormData] = useState({
  name: "",
  email: "",
  gender: "",
  hobbies: [],
  country: "",
});
```

- All form fields are stored in a single state object called formData.
- Each field corresponds to a key in the state (e.g., name, email, gender, hobbies, country).

Handling Input Changes:

- **const handleChange = (e) => { ... }**
  - handleChange updates the state whenever a user types in a text box, selects a radio button, checkbox, or dropdown.
  - Checkboxes are handled specially because multiple options can be selected, so we update the hobbies array in the state.

Form Fields:

1. **Text Input** (name)
2. **Email Input** (email)
3. **Radio Buttons** (gender) – allows selecting **one option**
4. **Checkboxes** (hobbies) – allows selecting **multiple options**
5. **Select Dropdown** (country) – allows choosing **one option**

Form Submission:

- **const handleSubmit = (e) => { ... }**
  - Prevents default HTML form submission using e.preventDefault().
  - Displays the collected form data in an alert (or can be sent to a backend API).

Why It's Useful

- React forms **collect and manage user input** dynamically.
- Using controlled components ensures the UI **stays in sync with state**.
- Useful for creating **registration forms, surveys, login forms**, and other interactive user inputs.

 # PROGRAM CODE

```
import React, { useState } from "react";

function App() {
 // State to store form data
 const [formData, setFormData] = useState({
   name: "",
   email: "",
   gender: "",
   hobbies: [],
   country: "",
 });

 // Options for hobbies
 const hobbyOptions = ["Reading", "Traveling", "Cooking", "Sports"];

 // Handle input changes
 const handleChange = (e) => {
   const { name, value, type, checked } = e.target;

   if (type === "checkbox") {
     let newHobbies = [...formData.hobbies];
     if (checked) {
       newHobbies.push(value);
     } else {
       newHobbies = newHobbies.filter((hobby) => hobby !== value);
     }
     setFormData({ ...formData, hobbies: newHobbies });
   } else {
```

```
    setFormData({ ...formData, [name]: value });
  }
};

// Handle form submission
const handleSubmit = (e) => {
  e.preventDefault();
  alert("Form Data Submitted:\n" + JSON.stringify(formData, null, 2));
};

return (
  <div style={{ maxWidth: "500px", margin: "50px auto" }}>
    <h2>React Form Example</h2>
    <form onSubmit={handleSubmit}>
      {/* Text Input */}
      <div>
        <label>Name:</label>
        <input
          type="text"
          name="name"
          value={formData.name}
          onChange={handleChange}
          required
        />
      </div>

      {/* Email Input */}
      <div>
        <label>Email:</label>
        <input
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          required
        />
```

```
</div>

{/* Radio Buttons */}
<div>
  <label>Gender:</label>
  <input
    type="radio"
    name="gender"
    value="Male"
    checked={formData.gender === "Male"}
    onChange={handleChange}
  />{" "}
  Male
  <input
    type="radio"
    name="gender"
    value="Female"
    checked={formData.gender === "Female"}
    onChange={handleChange}
  />{" "}
  Female
</div>

{/* Checkboxes */}
<div>
  <label>Hobbies:</label>
  {hobbyOptions.map((hobby) => (
    <label key={hobby} style={{ marginRight: "10px" }}>
      <input
        type="checkbox"
        name="hobbies"
        value={hobby}
        checked={formData.hobbies.includes(hobby)}
        onChange={handleChange}
      />
      {hobby}
```

```
          </label>
        ))}
      </div>

      {/* Select Dropdown */}
      <div>
        <label>Country:</label>
        <select
          name="country"
          value={formData.country}
          onChange={handleChange}
          required
        >
          <option value="">Select Country</option>
          <option value="India">India</option>
          <option value="USA">USA</option>
          <option value="UK">UK</option>
          <option value="Australia">Australia</option>
        </select>
      </div>

      {/* Submit Button */}
      <button type="submit" style={{ marginTop: "20px" }}>
        Submit
      </button>
    </form>
  </div>
 );
}

export default App;
```

Explanation

1. **Controlled Components**:
   - Each form field's value is controlled by React state (formData).
   - This allows React to keep track of form data dynamically.
2. **Text and Email Inputs**:
   - Standard <input> fields with type="text" and type="email".
3. **Radio Buttons (Gender)**:
   - Only one option can be selected at a time.
   - checked={formData.gender === "Male"} ensures the correct radio button is selected based on state.
4. **Checkboxes (Hobbies)**:
   - Multiple options can be selected.
   - The state array formData.hobbies stores all selected hobbies.
5. **Select Dropdown (Country)**:
   - <select> allows users to choose one option.
   - Controlled with value={formData.country}.
6. **Form Submission**:
   - handleSubmit prevents default form submission and displays an alert with form data.

Output

- User enters name and email, selects gender, hobbies, and country.
- On clicking **Submit**, the form displays:

Form Data Submitted:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "gender": "Male",
  "hobbies": ["Reading", "Sports"],
  "country": "USA"
}
```

PROGRAM-10

ReactJS – React Router, Updating the Screen

a. AIM: Write a program for routing to different pages using react router

What is React Router?

React Router is a standard library in React used for routing—i.e., navigating between different components (pages) in a single-page application (SPA) without reloading the page.

- It allows React apps to update the URL and render different components on the same page.

Key components:

1. <BrowserRouter> – Wraps the app to enable routing.
2. <Routes> – Contains all <Route> elements.
3. <Route> – Maps a URL path to a component.
4. <Link> – Used for navigation between routes without page reload.

- React Router is used to navigate between different pages/components in a React application.
- Using BrowserRouter, Routes, and Route, we can define paths and the components to render for eachpath.
- This example will show a simple 3-page application: Home, About, and Contact.

 # PROGRAM CODE

```
import React from "react";
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";

// Home Component
function Home() {
  return <h2>Welcome to the Home Page</h2>;
}

// About Component
function About() {
  return <h2>About Us Page</h2>;
}

// Contact Component
function Contact() {
  return <h2>Contact Us Page</h2>;
}

// App Component with Routing
function App() {
  return (
    <Router>
      <div style={{ textAlign: "center", marginTop: "50px" }}>
        <h1>React Router Example</h1>
```

```
      {/* Navigation Links */}
      <nav style={{ marginBottom: "20px" }}>
        <Link to="/" style={{ margin: "10px" }}>Home</Link>
        <Link to="/about" style={{ margin: "10px" }}>About</Link>
        <Link to="/contact" style={{ margin: "10px" }}>Contact</Link>
      </nav>

      {/* Routes */}
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </div>
  </Router>
 );
}

export default App;
```

Output:

Initial Render (Home Page)
React Router Example

[Home] [About] [Contact]

Welcome to the Home Page
The navigation links are at the top: Home | About | Contact.
Since the default path is /, the Home component is displayed.

After Clicking "About" Link
React Router Example

[Home] [About] [Contact]

About Us Page
The URL changes to /about.
The About component is rendered without reloading the page.

After Clicking "Contact" Link
React Router Example

[Home] [About] [Contact]

Contact Us Page
The URL changes to /contact.
The Contact component is rendered dynamically.

b. AIM: Write a program for updating the screen.

Updating the Screen

In React, updating the screen is mainly achieved through state changes. When the state of a component changes, React re-renders the component or parts of the UI that depend on that state.

 # PROGRAM for React Router and Screen Update

```
import React, { useState } from "react";
function App() {
  // useState hook to manage the count
  const [count, setCount] = useState(0);
return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Screen Update Example</h1>
      <p>Current Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase Count</button>
      <button onClick={() => setCount(count - 1)} style={{ marginLeft: "10px" }}>
        Decrease Count
      </button>
    </div>
  );
}
export default App;
```

Output

Initial Render:
Screen Update Example
Current Count: 0

[Increase Count] [Decrease Count]
After Clicking "Increase Count" Button Twice:
Screen Update Example
Current Count: 2

[Increase Count] [Decrease Count]
After Clicking "Decrease Count" Button Once:
Screen Update Example
Current Count: 1

[Increase Count] [Decrease Count]

PROGRAM-11

ReactJS – Hooks, Sharing data between Components

a. AIM: Write a program to understand the importance of using hooks.

Importance of Using Hooks

Hooks are special functions in React that let you **use state and other React features in functional components**, without writing class components.

This program demonstrates two essential hooks:

1. **useState**
   - Allows a functional component to maintain and update **state**.
   - In the program, it manages a **counter value** (count) that can be incremented or decremented by buttons.
2. **useEffect**
   - Allows a functional component to perform **side effects**, such as updating the browser title, fetching data, or subscribing to events.
   - In the program, it updates the **document title** whenever the count changes.

Key Takeaways:

- Hooks make functional components **powerful and stateful**, which previously required class components.
- They simplify code by avoiding **complex lifecycle methods** like componentDidMount and componentDidUpdate.
- Hooks improve **code readability, reusability, and maintainability**.

 # PROGRAM CODE

```
import React, { useState, useEffect } from "react";

function Counter() {
  // useState hook to manage count state
  const [count, setCount] = useState(0);

  // useEffect hook to perform side effects
  useEffect(() => {
    document.title = `You clicked ${count} times`;
    console.log(`Document title updated: ${count}`);
  }, [count]); // Dependency array: runs effect whenever 'count' changes

  return (
```

```
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>React Hooks Example</h1>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

export default Counter;
```

Explanation

1. **useState**:
   - Initializes count to 0.
   - setCount updates the count value.
2. **useEffect**:
   - Runs after every render where count changes.
   - Updates the browser tab title dynamically.
3. **Buttons**:
   - Clicking the buttons changes state, demonstrating how hooks manage state in functional components.

✅ **Importance of Hooks**

- Eliminates the need for class components for state and lifecycle methods.
- Makes code **more readable** and **less boilerplate**.
- Encourages **reusable logic** via custom hooks.

Output:

- Initial render:
  - Count: 0
  - Browser tab: You clicked 0 times
- Click **Increment**:
  - Count: 1
  - Browser tab: You clicked 1 times
- Click **Decrement**:
  - Count: 0
  - Browser tab: You clicked 0 times

b. AIM: Write a program for sharing data between components.

- In React, data flows from parent to child using props.
- This program will show a Parent component passing data to a Child component.
- We'll also show how a child can notify the parent using a function passed as a prop.

Program: Sharing Data Between Components

```
import React, { useState } from "react";

// Child Component
function Child({ message, sendDataToParent }) {
  return (
    <div style={{ border: "1px solid blue", padding: "10px", margin: "10px" }}>
      <h3>Child Component</h3>
      <p>Message from Parent: {message}</p>
      <button onClick={() => sendDataToParent("Hello Parent! From Child")}>
        Send Data to Parent
      </button>
    </div>
  );
}

// Parent Component
function Parent() {
  const [parentMessage, setParentMessage] = useState("Hi Child! From Parent");
  const [childData, setChildData] = useState("");

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Data Sharing Between Components</h1>

      <Child message={parentMessage} sendDataToParent={setChildData} />

      {childData && (
        <p style={{ marginTop: "20px", fontWeight: "bold" }}>
          Data received from Child: {childData}
        </p>
      )}
    </div>
  );
}

export default Parent;
```

Explanation

1. **Parent Component**
   - Holds the state parentMessage to send to the child.
   - Holds childData to store data received from the child.
2. **Child Component**
   - Receives message and sendDataToParent as props.
   - Displays the message from the parent.
   - Calls sendDataToParent function when the button is clicked to send data back to the parent.
3. **Data Flow**
   - **Parent → Child**: via props.message
   - **Child → Parent**: via props.sendDataToParent (callback function)

✅ **Key Points**

- Props are used for **sharing data from parent to child**.
- Functions passed as props allow **child-to-parent communication**.
- This is the **simplest and most common way** to share data between components in React.

OUTPUT:

Initial Render (Browser View)
Data Sharing Between Components

Child Component
Message from Parent: Hi Child! From Parent

[Send Data to Parent Button]
The child component displays the message received from the parent:
Hi Child! From Parent
There is a button labeled:
Send Data to Parent

After Clicking "Send Data to Parent" Button
Data Sharing Between Components

Child Component
Message from Parent: Hi Child! From Parent

[Send Data to Parent Button]
Data received from Child: Hello Parent! From Child
Clicking the button sends data from the child back to the parent.
The parent updates its state and displays:
Data received from Child: Hello Parent! From Child

PROGRAM – 12
MongoDB – Installation, Configuration, CRUD operations

a) AIM: Install MongoDB and configure ATLAS

MongoDB Installation

MongoDB is a **NoSQL, document-oriented database** that stores data in **JSON-like documents** instead of traditional rows and columns.

MongoDB can be used in two ways:

1. **Local Installation** – Installing MongoDB software on your system.
2. **Cloud-based Atlas** – Using MongoDB Atlas, a fully managed cloud database service that provides scalability, security, and easy access without manual setup.

Part 1: Install MongoDB (Locally)

On Windows
Download MongoDB
- ✓ Go to: **https://www.mongodb.com/try/download/community**
- ✓ Choose Windows, select the MSI installer.

Install MongoDB

- ✓ Run the installer.
- ✓ Select *Complete Installation*.
- ✓ Make sure *Install MongoDB as a Service* is checked.

Option 1: Use the Local System Account (Recommended) i.e, Install as a service (run MongoDB in background on your PC)
- During installation, when it asks **"Service Configuration"**, do the following:

- ✓ ☑ **Select**: Run service as Network Service or Local System
- ✓ ❌ **Do NOT select**: "Run service as a specific user"

This way, **you don't need to enter username/password**, and Windows will run the MongoDB service under a built-in system account.

a. Make sure that u have clicked the check-box below i.e;
☑ Install MongoDB Compass (GUI tool).

Verify the path of the folder where MongoDB is installing
Click on next→next->--to Finish.

Step-by-Step Fix: Add MongoDB to the PATH Environment Variable (Windows)

Step 1: Find MongoDB Installation Folder
1. Go to: C:\Program Files\MongoDB\Server\6.0\bin

The version (6.0) might be different on your system (e.g., 5.0, 4.4, etc.)
Make sure this folder has the files like mongo.exe, mongod.exe, etc.

Step 2: Add This Path to Environment Variables
1. Press Windows + S → Search for **Environment Variables**
2. Click **"Edit the system environment variables"**
3. In the System Properties window, click on **Environment Variables**

4. Under **System variables**, find and select the Path variable → Click **Edit**
5. Click **New** → Paste the MongoDB bin path:
6. C:\Program Files\MongoDB\Server\6.0\bin

Click **OK** on all windows to save.

Step 3: Verify Installation

Open *Command Prompt* and type:

mongod --version

- If installed correctly, it will show MongoDB version.

MongoDB Atlas

MongoDB **Atlas** is a **cloud-based, fully managed database service** for MongoDB. It runs on **AWS, Azure, and Google Cloud**, allowing you to create, run, and scale MongoDB databases without managing servers yourself.

☑ **Key points**

- Hosted MongoDB in the cloud.
- Fully managed (backup, scaling, monitoring, security).
- Runs on AWS, Azure, GCP.
- Provides free and paid clusters.
- Easy to connect with apps using a connection string.

Configure MongoDB Atlas (Cloud)

1. **Create Atlas Account**
   - Go to **https://www.mongodb.com/cloud/atlas**
   - Sign up (use Google/GitHub/email).
2. **Create a Cluster**
   - Click *Build a Database*.
   - Choose a free tier (M0 – free, 512MB storage).
   - Pick a cloud provider (AWS, Azure, or GCP).
   - Select a region (closest to you).
   - Click *Create Cluster*.
3. **Create a Database User**
   - Go to *Database Access*.
   - Add new user (set username & password).
   - Save credentials.
4. **Configure Network Access**
   - Go to *Network Access*.

- Add IP address → choose *Allow Access from Anywhere* (0.0.0.0/0).
- Or add your local machine IP for security.

5. **Connect to Cluster**
   - Click *Connect → Connect your application*.
   - Get the **Connection String** from the "Connect" option.

     Example:

     mongodb+srv://username:password@cluster0.abcd.mongodb.net/myDatabase
6. **Connect using MongoDB Shell**, Compass, or directly from applications.

   mongosh "mongodb+srv://cluster0.abcd.mongodb.net/myDatabase" --username myUser

Benefits of MongoDB Atlas

- No need to install or maintain servers.
- Provides automatic scaling and backups.
- Accessible from anywhere with an internet connection.
- Secure access control with authentication and IP whitelisting.

☑ Now you have both:

- Local MongoDB installed.
- Atlas configured and ready to connect from your app.

MongoDB Atlas Vs MongoDB Compass

MongoDB Atlas	**MongoDB Compass**
Hosts your database in the cloud	Lets you view & manage a database
Handles server maintenance	GUI tool for human interaction
Can run MongoDB without installing locally	Requires a MongoDB server to connect to
Cloud service with replication & backup	Desktop app for browsing/querying

☑ **Summary:**

- Use **Atlas** if you want a **managed MongoDB server in the cloud**.
- Use **Compass** if you want a **GUI to explore, query, and manage MongoDB databases** (Atlas or local).
- You can use them together: **Atlas hosts your database → Compass connects to it for easy management**.

b. AIM: Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove().

CRUD stands for **Create, Read, Update, and Delete**, which are the basic operations to manage data in a MongoDB collection. MongoDB provides built-in methods like insert(), find(), update(), and remove() to perform these operations.

Working with MongoDB queries in MongoDB compass (GUI Tool)

- Go to MongoDB compass Icon on the Desktop & open it.
- Click on "Add New Connection"
- Verify that it is displaying MongoDB URL with the port number connected.
- Give your preferred name & color and click on "Save & Connect".
- It will open the new connection window and click on any name of the database displayed
- It will show the option "open MongoDB command shell" click on it.
- The command shell for MongoDB is opened to work by opening with default database i.e. **test**

Example of working with Mongo DB queries:

Command: show databases (or shorthand **show dbs)**
-→
admin 40.00 KiB
config 108.00 KiB
local 72.00 KiB

- ✓ used to list all the databases present on the MongoDB server.
- ✓ When you run show databases (or shorthand show dbs) in the Mongo Shell, it displays a list of all the available databases on the server.
- ✓ Each database is shown along with its size on disk.
- ✓ A database will only appear in the list if it has at least one collection with some data. (Empty databases are not shown.)

Command: use school

---→ switched to db school

- ✓ The use command tells MongoDB shell which database you want to work with.
- ✓ If the database already exists, MongoDB switches to it.
- ✓ If the database does not exist yet, MongoDB creates it only after you insert data (like creating a collection or document).

A **collection in MongoDB is a container for documents, similar to a table in SQL.**

Key points:

- A **document** = one record (stored in JSON-like format, BSON).
- A **collection** = multiple related documents stored together.
- Unlike SQL tables, collections **don't require a fixed schema** → each document can have different fields.
- **Example:**

- { "name": "Raju", "age": 20 }
- { "name": "Sita", "class": "10th", "marks": 450 }

Both can exist in the same collection, even though fields differ

Use db.createCollection() → manually create an empty collection.

// Creating Collections

Command: db.createCollection('class10')

{ ok: 1 }

Command: db.createCollection('class9')

{ ok: 1 }

Command: show collections

used to list all the collections present in the current database.

→
class10
class9

1. Create (Insert Documents)

The insert() method is used to add new documents into a collection.

MongoDB provides different methods for inserting data:

1. **insertOne()** → Insert a single document
2. **insertMany()** → Insert multiple documents at once
3. **insert()** → Older version (can insert single/multiple, but now replaced by the above methods)

Command: db.class10.insertOne({'name':'Raju', 'rollno':1})

→
```
{
  acknowledged: true,
  insertedId: ObjectId('68b2b3c0f1189b6ebcbd4420')
}
```

Command : db.class10.insertOne({'name':'ramu','rollno':2,'phone':123456789})

→
```
{
  acknowledged: true,
  insertedId: ObjectId('68b2b48ef1189b6ebcbd4421')
}
```

Command: db.class10.insertOne({'name':'ramu','rollno':3})

→
```
{
  acknowledged: true,
  insertedId: ObjectId('68b2b722f1189b6ebcbd4422')
}
```

2. Read (Find Documents)

The find() method is used to retrieve documents from a collection.

What is find()?

- The **find()** method in MongoDB is used to **retrieve documents** (records) from a **collection**.
- By default, it returns **all documents** in the collection.
- You can also pass conditions (filters), projection (fields to show), sorting, and limits.

db.collection.find(query)
query → criteria/conditions to match documents (like WHERE in SQL).

// Find all documents

Command: db.class10.find()

→
```
{
  _id: ObjectId('68b2b3c0f1189b6ebcbd4420'),
  name: 'Raju',
  rollno: 1
}
{
  _id: ObjectId('68b2b48ef1189b6ebcbd4421'),
  name: 'ramu',
  rollno: 2,
  phone: 123456789
}

{
  _id: ObjectId('68b2b722f1189b6ebcbd4422'),
  name: 'ramu',
  rollno: 3
}
```

Command: show dbs

```
admin    40.00 KiB
config  108.00 KiB
local    72.00 KiB
mydb     72.00 KiB
school   80.00 KiB
```

// Find with a filter

- **db.class10.find({'rollno':2})**

// gives only one value

- **db.class10.findOne({'rollno':2})**

```
{
  _id: ObjectId('68b2b48ef1189b6ebcbd4421'),
  name: 'ramu',
  rollno: 2,
  phone: 123456789
}
```

Command: db.class10.find()

```
 {
_id: ObjectId('68b2b48ef1189b6ebcbd4421'),
 name: 'ramu',
 rollno: 2,
 phone: 123456789
}
{
 _id: ObjectId('68b2b722f1189b6ebcbd4422'),
 name: 'ramu',
 rollno: 3
}
```

Command: db.class10.find({'name':'ramu'})

```
{
id: ObjectId('68b2b48ef1189b6ebcbd4421'),
 name: 'ramu',
 rollno: 2,
 phone: 123456789
}

{
 _id: ObjectId('68b2b722f1189b6ebcbd4422'),
 name: 'ramu',
 rollno: 3

}
```

3. Update (Modify Documents)

What is Update?

- The **update methods** in MongoDB are used to **modify existing documents** in a collection.
- Unlike insert (which adds new documents), update changes fields of documents that already exist.

◇ **Main Update Methods**

1. **updateOne()** → updates the **first matching document**.
2. **updateMany()** → updates **all matching documents**.
3. **replaceOne()** → replaces the entire document with a new one.
4. (Older method: update(), now mostly replaced by the above.)

Command: db.class10.updateOne({'rollno':2}, { $set: { 'phone': 1234567890 } })

Finds the document where **rollno = 2** and updates **phone to 1234567890**

4. Delete (Remove Documents)
The remove() method is used to delete documents from a collection.

Main Delete Methods

1. **deleteOne()** → removes the **first matching document**.
2. **deleteMany()** → removes **all documents** matching a condition.
3. (Old method: remove(), now replaced by the above two).

Command: db.class10.deleteOne({'name':'Raju'})

PROGRAM -13

MongoDB – Databases, Collections and Records

a) AIM: Write MongoDB queries to Create and drop databases and collections.

In MongoDB, databases and collections are the fundamental storage structures. Databases act as containers for collections, while collections store documents (records). To manage them, MongoDB provides specific commands:

1. **Creating a Database**
   - MongoDB does not require an explicit CREATE DATABASE command.
   - A database is created automatically when you switch to it using the use command and insert at least one document.
   - MongoDB creates a database when you switch to it and insert at least one document.

// Switch to (or create) a database

use myDatabase

Output:

switched to db myDatabase

2. **Drop a Database**

To delete the currently active database:

// Drops the selected database

db.dropDatabase()

Output:

{ "dropped" : "myDatabase", "ok" : 1 }

3. **Create a Collection**

Collections are created either automatically when a document is inserted, or explicitly using createCollection().

// Explicitly create a collection

db.createCollection("students")

Output:

{ "ok" : 1 }

4. Drop a Collection

To remove a collection and all its documents:

// Drop the "students" collection

db.students.drop()

Output:

true

☑ **Summary:**

- Created a new database using use.
- **use <dbName>** → Creates/switches to a database.
- Created a collection with db.createCollection().
- **db.createCollection("name")** → Creates a collection.
- Verified collections with **show collections.**
- Dropped a collection using .drop().
- **db.collectionName.drop()** → Drops a collection.

Deleted the entire database using db.dropDatabase().

- **db.dropDatabase()** → Deletes a database.

❖ **Important Commands**

use mydb	-- switch to DB (doesn't create until write)
db.createCollection("c")	-- create empty collection (explicit)
db.c.insertOne({...})	-- insert → creates collection if needed
show dbs	-- list databases (only non-empty ones)
show collections	-- list collections in current DB
db.c.drop()	-- drop collection
db.dropDatabase()	-- drop current database
db.collection.deleteMany({})	-- remove all docs but keep collection

b)Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

In MongoDB, records are stored as documents inside collections. To query and manipulate these records, several methods are commonly used:

Step 1: Retrieve Records using find()

Task: Fetch all records from the students collection.

Query:

db.students.find()

Expected Output (sample):

{ "_id": 1, "name": "Ravi", "department": "CSE" }

{ "_id": 2, "name": "Anita", "department": "CSE" }

Step 2: Apply Conditions with find()

Task: Fetch only students from the CSE department.

Query:

db.students.find({ "department": "CSE" })

Expected Output (sample):

{ "_id": 3, "name": "Sita", "department": "CSE" }

{ "_id": 4, "name": "Kiran", "department": "CSE" }

Step 3: Limit Results using limit()

Task: Display only the first 3 records.

Query:

db.students.find().limit(3)

Expected Output:
(Only 3 student documents shown, even if more exist in collection)

Step 4: Sort Records using sort()

Task: Sort students by name in ascending order.

Query:

db.students.find().sort({ "name": 1 })

Expected Output (sample):

{ "_id": 7, "name": "Anita", "department": "ECE" }

{ "_id": 2, "name": "Kiran", "department": "CSE" }

{ "_id": 4, "name": "Ravi", "department": "CSE" }

Step 5: Create Index using createIndex()

Task: Create an index on the name field to speed up searches.

Query:

db.students.createIndex({ "name": 1 })

Expected Output:

"name_1"

Step 6: Aggregate Records using aggregate()

Task: Count the number of students in each department.

Query:

```
db.students.aggregate([
  { $group: { _id: "$department", total: { $sum: 1 } } }
])
```

Expected Output (sample):

{ "_id": "CSE", "total": 25 }

{ "_id": "ECE", "total": 18 }

☑ **Conclusion:**

☑ **Summary of Commands:**

- find() → Retrieve records.
- limit() → Restrict number of results.
- sort() → Order results.
- createIndex() → Improve query performance.
- aggregate() → Perform advanced computations like grouping, counting, summing.

COMPUTER NETWORKS
LAB MANUAL

B. TECH
III YEAR – I SEM (R23)
(2025-26)

DEPARTMENT OF CSE-AIML

Aditya College of Engineering & Technology
Aditya Nagar, ADB Road, Surampalem – 533437

Syllabus

1. Study network devices in detail and connect computers within a Local Area Network (LAN).
2. Write a program to implement data link layer framing methods, including:

   i) Character stuffing

   ii) Bit stuffing
3. Develop a program to implement the checksum method for error detection in data link layer framing.
4. Write a program for generating Hamming codes for error detection and correction.
5. Implement programs for three Cyclic Redundancy Check (CRC) polynomials: CRC-12, CRC-16, and CRC-CCIP on a given data set of characters.
6. Write a program to implement the Sliding Window protocol for Go-Back-N ARQ.
7. Write a program to implement the Sliding Window protocol for Selective Repeat ARQ.
8. Develop a program to implement the Stop-and-Wait protocol.
9. Write a program to demonstrate congestion control using the Leaky Bucket algorithm.
10. Implement Dijkstra's algorithm to compute the shortest path in a graph.

COMPUTER NETWORKS LAB

III B.Tech I-Semester

COURSE OUTCOMES

S.No	Course Code - CO	Course Outcomes	Blooms Taxonomy
1	CO1	***Know*** how to connect computers in LAN and ***implement*** different framing methods.	Application
2	CO2	***Implement*** error detection correction techniques.	Application
3	CO3	***Know*** how reliable data communication is achieved through data link layer.	Application
4	CO4	***Suggest*** appropriate routing algorithm for the network.	Application
5	CO5	***Provide*** internet connection to the system and its installation.	Application
6	CO6	***Work*** on various network management tools	Application

COMPUTER NETWORKS LAB

III B.Tech I-Semester

CO &POs MAPPING

Course Code	Course Outcomes	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12
CO1	***Know*** how to connect computers in LAN and ***implement*** different framing methods.	2	2	1	1	1				1			1
CO2	***Implement*** error detection correction techniques.	2	2	3	1	1				2			2
CO3	***Know*** how reliable data communication is achieved through data link layer.	1	2	2	2	1				1			2
CO4	***Suggest*** appropriate routing algorithm for the network.	2	2	2	2	1				2			2
CO5	***Provide*** internet connection to the system and its installation.	1	2	2	2	1				1			2
CO6	***Work*** on various network management tools	1	2	2	2	2				1			2

COMPUTER NETWORKS LAB (R2032121)

III B.Tech I-Semester

CO & PSO MAPPING

Course Code	Course Outcomes	PSO1	PSO2	PSO3
CO1	***Know*** how to connect computers in LAN and ***implement*** different framing methods.	2	2	1
CO2	***Implement*** error detection correction techniques.	2	2	3
CO3	***Know*** how reliable data communication is achieved through data link layer.	1	2	2
CO4	***Suggest*** appropriate routing algorithm for the network.	2	2	3
CO5	***Provide*** internet connection to the system and its installation.	1	2	3
CO6	***Work*** on various network management tools	1	2	3

GENERAL INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.

2. Student should enter into the laboratory with:

   - Laboratory observation notes.
   - Laboratory Record updated up to the last session experiments.
   - Proper Dress code and Identity card.

3. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

4. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

5. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

6. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.

7. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

8. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

9. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

TOOLS USED DURING THE LAB

PYTHON	python™
WIRESHARK	Wireshark
NMAP	NMAP
NS2 SIMULATOR	ns-2 NETWORK SIMULATOR

INDEX

EXPERIMENT 1

Study of Network devices in detail and connect the computers in Local Area Network.

AIM: Study of various network devices in detail All but the most basic of networks require devices to provide connectivity and functionality.Understanding how these networking devices operate and identifying the functions they performare essential skills for any network administrator and requirements for aNetwork+ candidate.Theall networkdevicesareexplained below:

Hubs:

The hub or network hub connects computers and devices and sends messages and data from any one device to all the others.If the desktop computer wants to send data to the laptop and it sends amessage to the laptop through the hub,the message willgetsentby the hub to all the computers and devices on the network.They need to do work to figure out that the message is not for them. The message also uses up bandwidth (room) on the network wires or wireless radiowavesandlimits howmuch communication can go on.Hubs arenot usedoften thesedays.

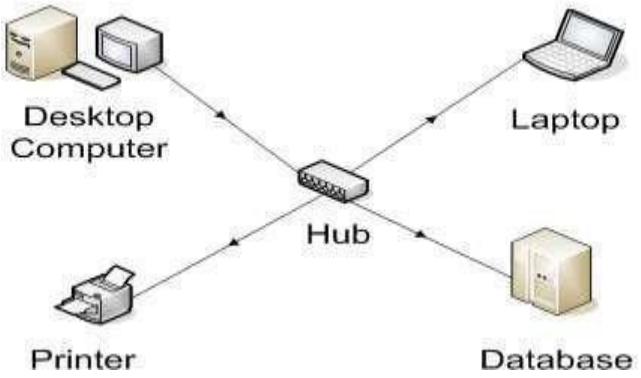


Figure 1

Switch:

The switch connects the computer network components but it is smart about it. It knows the address of each item and so when the desktop computer wants to talk to the laptop, it only sends the message to the laptop and nothing else. In order to have a small home network that just connects the local equipment all that is really needed is a switch and network cable or the switch can transmit wireless information that is received by wireless receivers that each of the network devices have.

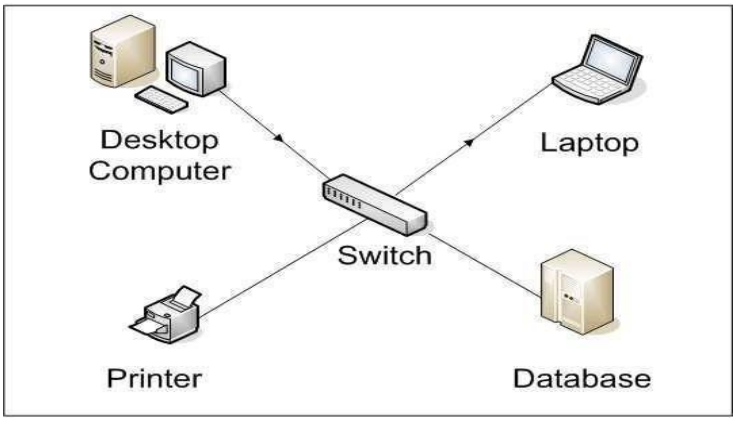


Figure 2

Bridges:

Bridges are used to divide larger networks into smaller sections. They do this by sitting between two physical network segments and managing the flow of data between the two. By looking at the MAC address of the devices connected to each segment, bridges can elect to forward the data (if they believe that the destination address is on another interface), or block it from crossing (if they can verify that it is on the interface from which it came).

A bridge functions by blocking or forwarding data, based on the destination MAC address written into each frame of data. If the bridge believes the destination address is on a network other than that from which the data was received, it can forward the data to the other networks to which it is connected. If the address is not on the other side of the bridge, the data is blocked from passing. Bridges "learn" the MAC addresses of devices on connected networks by "listening" to network traffic and recording the network from which the traffic originates. Figure 3 shows a representation of a bridge.

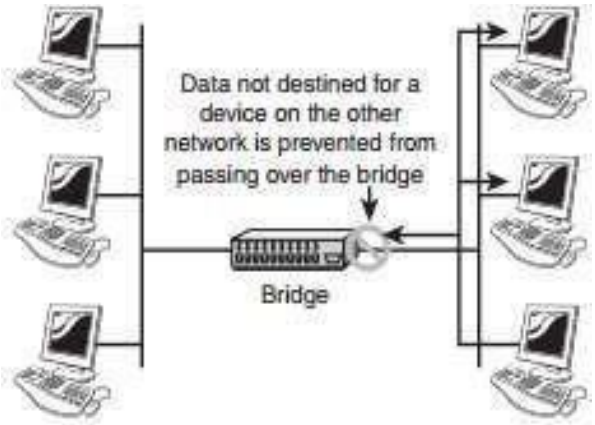


Figure 3

Routers:

In a common configuration, routers are used to create larger networks by joining two network segments. A router derives its name from the fact that it can route data it receives from one network onto another. When a router receives a packet of data, it reads the header of the packet to determine the destination address. Once it has determined the address, it looks in its routing table to determine whether it knows how to reach the destination and, if it does, it forwards the packet to the next hop on the route. The next hop might be the final destination, or it might be another router. Figure 4 shows, in basic terms, how a router works.

The routing tables play a very important role in the routing process. They are the means by which the router makes its decisions. For this reason, a routing table needs to be two things. It must be up-to-date, and it must be complete. There are two ways that the router can get the information for the routing table—through static routing or dynamic routing.

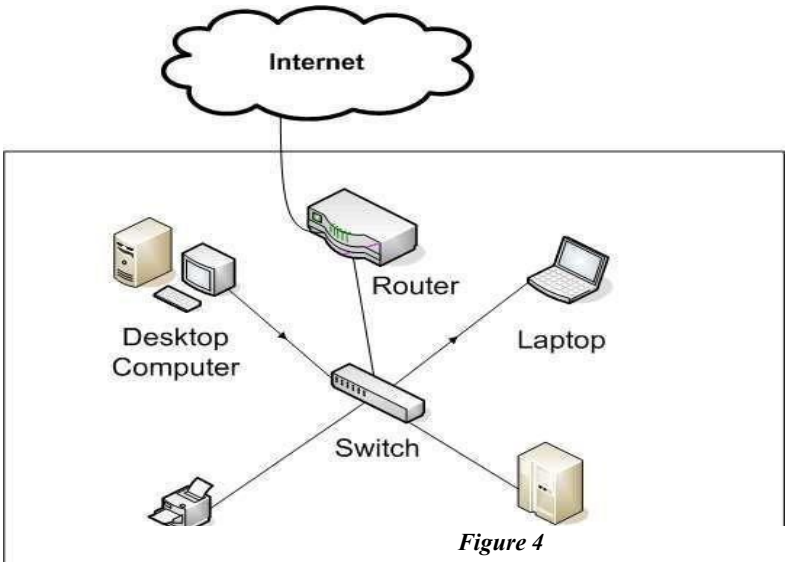


Figure 4

Modem:

Most everyone wants to connect to the internet. A broadband modem is used to take a high speed Internet connection provided by an ISP (Internet Service Provider) and convert the data into a form that your local network can use. The high speed connection can be DSL (Digital Subscriber Line) from a phone company or cable from a cable television provider.

In order to be reached on the Internet, your computer needs a unique address on the internet. Your ISP will provide this to you as part of your Internet connection package. This address will generally not be fixed which means that they may change your address from time to time. For the vast majority of users, this makes no difference. If you have only one computer and want to connect to the Internet, you strictly speaking don't need a router. You can plug the network cable from the modem directly into the network connection of your computer. However, you are much better off connecting the modem to a router. The ip address your ISP provides will be assigned to the router.

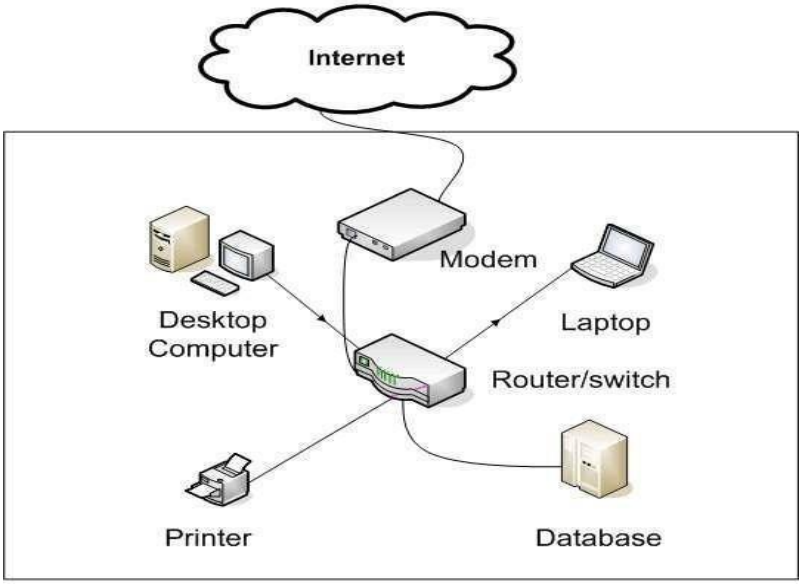


Figure 5

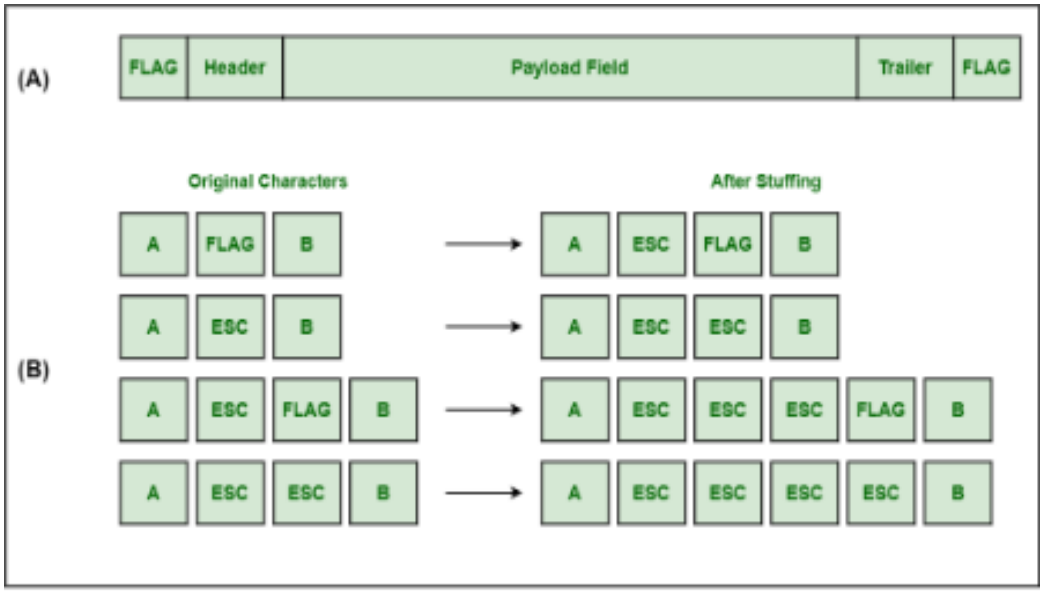
EXPERIMENT 2

2. i) Write a Program to implement the data link layer framing methods such as --> character stuffing.

Character Stuffing :

Character stuffing is also known as byte stuffing or character-oriented framing and is same as that of bit stuffing but byte stuffing actually operates on bytes whereas bit stuffing operates on bits. In byte stuffing, special byte that is basically known as ESC (Escape Character) that has predefined pattern is generally added to data section of the data stream or frame when there is message or character that has same pattern as that of flag byte.

But receiver removes this ESC and keeps data part that causes some problems or issues. In simple words, we can say that character stuffing is addition of 1 additional byte if there is presence of ESC or flag in text



A Character Stuffing

(A) A frame delimited by flag bytes

(B) Four examples of byte sequences before and after byte stuffing

PROGRAM:

```
head = input("Enter character that represents the starting
delimiter: ")
tail = input(" Enter character that represents the ending
delimiter: ")
st = input("Enter the characters to be stuffed: ")
res=head
for i in st:
      if i==head or i ==tail:
            res = res + i + i
      else:
            res = res + i
res = res+tail
print("Frame after character stuffing: ", res)
```

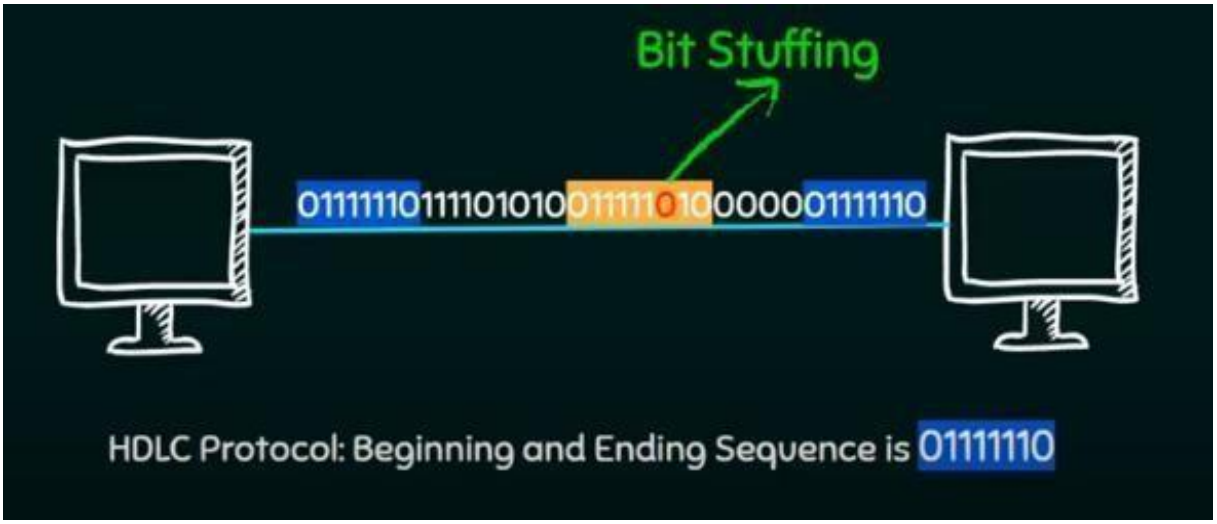
OUTPUT:

```
Enter character that represents the starting delimiter: d

Enter character that represents the ending delimiter: g

Enter the characters to be stuffed: goodday

Frame after character stuffing: dggoodddayg
```

2. ii) Write a Program to implement the data link layer framing methods such as --> bit stuffing.

BitStuffing :

Bit stuffing is also known as bit-oriented framing or bit-oriented approach. In bit stuffing, extra bits are being added by network protocol designers to data streams. It is generally insertion or addition of extra bits into transmission unit or message to be transmitted as simple way to provide and give signaling information and data to receiver and to avoid or ignore appearance of unintended or unnecessary control sequences.



PROGRAM:

```
st = input ("Enter the frame: ") count = 0
res = ""
for i in st:
       if i == '1' and count < 5:
           res += '1'
           count += 1
       elif i == ' ':
           pass
       else:
           res += i count = 0
       if count == 5:
           res += '0'
           count= 0
print("Frame after bit stuffing: ", res)
```

OUTPUT:

```
Enter the frame: 01111110

Frame after bit stuffing: 011111010
```

EXPERIMENT 3

3. Write a Program to implement data link layer framing method checksum.

PROGRAM:

```
s1 = input("Enter the string of 0's and 1's as subunit1: ")
s2 = input("Enter the string of 0's and 1's as subunit2: ")
# Reverse both strings for easier addition (LSB first)
s1 = s1[::-1]
s2 = s2[::-1]
res = ""
c = '0'   # carry
# Binary addition of s1 and s2
for i, j in zip(s1, s2):
    if i == '0' and j == '0' and c == '0':
        res += '0'
        c = '0'

    elif i == '0' and j == '0' and c == '1':
        res += '1'
        c = '0'

    elif i == '0' and j == '1' and c == '0':
        res += '1'
        c = '0'

    elif i == '0' and j == '1' and c == '1':
        res += '0'
        c = '1'
```

```
        elif i == '1' and j == '0' and c == '0':
            res += '1'
            c = '0'

        elif i == '1' and j == '0' and c == '1':
            res += '0'
            c = '1'

        elif i == '1' and j == '1' and c == '0':
            res += '0'
            c = '1'

        elif i == '1' and j == '1' and c == '1':
            res += '1'
            c = '1'
    # Handle final carry
    if c == '1':
        ans = ""
        for i in res:
            if i == '1' and c == '1':
                ans += '0'
                c = '1'
            elif i == '0' and c == '0':
                ans += '0'
                c = '0'
            else:
                ans += '1'
                c = '0'
        res = ans
```

```
# Take 1's complement
final = ""
for i in res:
    if i == '1':
        final += '0'
    else:
        final += '1'

print("Checksum of two subunits: ", final[::-1].strip())
```

OUTPUT:

```
Enter the string of 0's and 1's as subunit1: 10101001
Enter the string of 0's and 1's as subunit2: 00111001
Checksum of two subunits: 00011101
```

EXPERIMENT 4

4. Write a program for Hamming Code generation for error detection and correction.

PROGRAM:

```
li = list(map(int,input("Enter 7 bits data of 0's and 1's separated
by spaces: ").split()))
rec = list(map(int,input("Enter the received 11 data bits of 0's
and 1's separated by spaces: ").split()))
# reverse the list
li = li[::-1]
# parity bits of 0 are added at the place of 2 pow's i.e. at
positions of 1,2,4,8 remaining places data bits are added
li = [0,0] + li[0:1] + [0] + li[1:4] + [0] + li[4:]
#now find the even parity bit position
li[0] = (li[2] + li[4] + li[6] + li[8] + li[10]) % 2
li[1] = (li[2] + li[5] + li[6] + li[9] + li[10]) % 2
li[3] = (li[4] + li[5] + li[6]) % 2
li[7] = (li[8] + li[9] + li[10]) % 2
# reverse the list
li = li[::-1]
#reverse the receiver side data and check the parity bits position
values
rec = rec[::-1]
r1 = (rec[0] + rec[2] + rec[4] + rec[6] + rec[8] + rec[10]) % 2
r2 = (rec[1] + rec[2] + rec[5] + rec[6] + rec[9] + rec[10]) % 2
r3 = (rec[3] + rec[4] + rec[5] + rec[6]) % 2
r4 = (rec[7] + rec[8] + rec[9] + rec[10]) % 2

bit = str(r4) + str(r3) + str(r2) + str(r1)
bit = int(bit,2)
if bit :
   print("received data is having error at position: ", bit)
else:
   print("received data doesn't have any error")
```

OUTPUT:

```
Enter 7 bits data of 0’s and 1’s separated by spaces: 1 0 1 0
1 0 1

Enter the received 11 data bits of 0's and 1's separated by
spaces 1 0 1 0 0 1 0 1 1 0 1

Received data is having error at position: 2

Enter 7 bits data of 0's and 1's separated by spaces: 1 0 1 0
1 0  1

Enter the received 11 data bits of 0's and 1's separated by 1
1 1 spaces:    1 0 1 0 0 1 0 1

received data doesn't have any error
```

EXPERIMENT 5

5. Write a Program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCITT.

PROGRAM

```
def xor(x, y):
    ans = ""
    for i in range(len(y)): # Compare all bits
        if x[i] == y[i]:
            ans += '0'
        else:
            ans += '1'
    return ans


def divide(dividend, divisor):
    a = len(divisor)
    temp = dividend[0:a]
    # Loop until all bits of the dividend have been processed
    while a < len(dividend):
        if temp[0] == '1':
            temp = xor(divisor, temp) + dividend[a]
        else:
            temp = xor('0' * a, temp) + dividend[a]
        a += 1
        if temp[0] == '1':
            temp = xor(divisor, temp)
        else:
            temp = xor('0' * a, temp)
    return temp
```

```
# Predefined generator polynomials
keys = ['1100000001111','11000000000000101',
  '10001000000100001']
print("Choose the CRC")
print("1. CRC - 12")
print("2. CRC - 16")
print("3. CRC - CCITT ")

n = int(input("Enter your choice (1/2/3): "))

send = input("Enter the string of binary data bits to be sent
from the sender: ")
rec = input("Enter the string of binary data received at the
receiver side: ")

# Select the appropriate key
key = keys[n - 1]

# Encoding on sender's side
length = len(key)
send1 = send + '0' * (length - 1)
rem = divide(send1, key)
# Decoding on receiver's side
ans = divide(rec, key)
# Check for transmission errors
if ans == '0' * (len(key) - 1):
    print("No error in transmission ")
else:
    print("Frame error detected ")
```

OUTPUT:

```
Choose the CRC


1. CRC - 12

2. CRC- 16

3. CRC- CCITT

1

Enter the string of code word of binary data bits of 0's and
1's to be sent from the sender: 1011


Enter the string of code word of binary data received at the
receiver side: 1011110

Sent Codeword: 1011001

no error


Choose the CRC


1. CRC - 12

2. CRC- 16

3. CRC- CCITT

2

Enter the string of code word of binary data bits of 0's and
1's to be sent from the sender: 101110111010101

Enter the string of code word of binary data received at the
receiver side: 10111011101010101001100111111011

no error


Choose the CRC


1. CRC- 12
```

```
2. CRC- 16

3. CRC- CCITT

1


Enter the string of code word of binary data bits of 0's and
1's to be sent from the sender: 1010101

Enter the string of code word of binary data received at the
receiver side: 1010101001000000010


no error
```

EXPERIMENT 6

6. Write a Program to implement Sliding window protocol for Go back N.

PROGRAM:

SENDER SIDE:

```
import socket
import random
import time

s = socket.socket()
s.bind(("localhost", 1450))
s.listen(5)
c, adr = s.accept()
print(str(adr))
n = int(input("Enter number of frames: "))
N = int(input("Enter window size: "))
seq = 1 # is used to keep track of the window starting
frame = 1 # frame to send starts with 1
# send first N window size frames
for i in range(N):
      print('Frames sent ->', frame)
      c.send(str(frame).encode())
      frame += 1
      time.sleep(2)
 timer = 5

# will start with acknowledgement frame of 1
while frame <= n:
```

```
 t = random.randint(1, 7)

 msg = c.recv(1).decode()

 msg = int(msg)


if (msg != seq):

# here we try to discard the already sent frames after
failed frame

      continue

if (timer > t):

# if the timer is greater than random number be consider
it as ack

      print("acknowledgement received")

      print('Frames sent ->', str(frame))

      # we will send next frame

      c.send(str(frame).encode())

      seq += 1

      frame += 1

      time.sleep(2)

else:

# if timer is less than the random number we consider as
not received ack

      print('acknowledgement not received')

      frame = seq

       #  we  will  again  send  the  frames  from  window
       starting i.e. seq

      for i in range(N):

           print('Frames sent ->', frame)

           c.send(str(frame).encode())

           frame += 1

           time.sleep(2)
```

RECEIVER SIDE:

```
import socket
import time
s=socket.socket()
s.connect(("localhost", 1450))
while 1:
      msg=s.recv(2).decode()
      print("Received --> ",int(msg))
      s.send(str(msg).encode())
      time.sleep(1)
```

OUTPUT:

SENDER SIDE:

Enter number of frames: 8

Enter window size: 4

Frames sent -> 1

Frames sent -> 2

Frames sent -> 3

Frames sent -> 4

acknowledgement received

Frames sent -> 5

acknowledgement received

Frames sent -> 6

acknowledgement not received

RECEIVER SIDE:

Received --> 1

Received --> 2

Received --> 3

Received --> 4

Received --> 5

Received --> 6

Received --> 3

Received --> 4

Received --> 5

Received --> 6

Received --> 3

Frames sent -> 3	Received --> 4
Frames sent -> 4	Received --> 5
Frames sent -> 5	Received --> 6
Frames sent -> 6	Received --> 7
acknowledgement not received	Received --> 8
Frames sent -> 3	
Frames sent -> 4	
Frames sent -> 5	
Frames sent -> 6	
acknowledgement received	
Frames sent -> 7	
acknowledgement received	
Frames sent -> 8	

EXPERIMENT 7

7. Write a Program to implement Sliding window protocol for Selective repeat.

PROGRAM:

SENDER SIDE:

```
import socket
import random
import time
s = socket.socket()
s.bind(("localhost",8038))
s.listen(5)
c, adr = s.accept()
print("from address”, str(adr), "connection has established")
n = int(input("Enter number of frames: "))
N = int(input("Enter window size: "))
seq = 1 # is used to keep track of the window starting
frame = 1 # frame to send starts with 1
# send first N window size frames
for i in range(N):
    print('Frames sent ->',frame)
    c.send(str(frame).encode()) frame += 1
    time.sleep(2)
timer = 5 # will start with acknowledgement frame of 1
while frame <= n :
     t = random.randint(1,7)
     msg = c.recv(1).decode()
```

```
        msg = int(msg)
        print("Frame ", msg)
        if(timer > t):
            print("acknowledgement received")
            print('Frames sent ->',str(frame))
            c.send(str(frame).encode())
            seq += 1
            frame += 1
            time.sleep(2)
        else:
            print('acknowledgement not received')
            print('Frames sent ->',msg)
            c.send(str(msg).encode())
            time.sleep(2)
```

RECEIVER SIDE:

```
import socket
import time
s=socket.socket()
s.connect(("localhost", 8038))
while 1:
   msg=s.recv(2).decode()
   print("Received --> ",int(msg))
   s.send(str(msg).encode())
   time.sleep(1)
```

OUTPUT:

SENDER SIDE:

Enter number of frames: 8

Enter window size: 4

Frames sent -> 1

Frames sent -> 2

Frames sent -> 3

Frames sent -> 4

Frame 1

acknowledgement not received

Frames sent -> 1

Frame 2

acknowledgement received

Frames sent -> 5
Frame 3
acknowledgement received
Frames sent -> 6
Frame 4
acknowledgement received
Frames sent -> 7
Frame 1
acknowledgement not received
Frames sent -> 1
Frame 5
acknowledgement received
Frames sent -> 8
Received --> 7
Received --> 1
Received --> 8

RECEIVER SIDE:

Received --> 1

Received --> 2

Received --> 3

Received --> 4

Received --> 1

Received --> 5

Received --> 6

Received --> 7

Received --> 1

Received --> 8

Received --> 1

EXPERIMENT 8

8. Write a Program to implement Stop and Wait Protocol.

PROGRAM:

SENDER SIDE:

```
import socket
import time
import random
s=socket.socket()
s.bind(("localhost", 8020))
s.listen(5)
c, adr = s.accept()
print("connection to " + str(adr) + " established")
a=int(input("enter total number of frames"))
x = 0
print("sending -->", x)
c.send(str(x).encode())
while( a > 1 ):
     timer = 5
     t=random.randint(1,7)
     msg = c.recv(1).decode()
     if (timer > t):
        time.sleep(3)
        print("ack-->", msg)
        x=int(msg)
        print("sending -->", str(x))
        c.send(str(x).encode())
```

```
    else:
            time.sleep(3)
            print("timeout")
            print("sending again-->", x)
            c.send(str(x).encode())
            a=a+1
    a = a-1
```

RECEIVER SIDE:

```
import socket
s=socket.socket()
s.connect(("localhost", 8020))
while(1):
  msg=s.recv(1).decode()
  print("Received --> ", msg)
  x=int(msg)
  if(x==0):
     x=x+1
     s.send(str(x).encode())
  else:
     x=x-1
     s.send(str(x).encode())
```

OUTPUT:

SENDER SIDE:

enter total number of frames6

sending --> 0

timeout

sending again--> 0

timeout

sending again--> 0

ack--> 1

sending --> 1

ack--> 0

sending --> 0

timeout

sending again--> 0

timeout

sending again--> 0

timeout

sending again--> 0

ack--> 1

sending --> 1

timeout

sending again--> 1

timeout

sending again--> 1

ack--> 0

sending --> 0

RECEIVER SIDE:

Received --> 0

Received --> 0

Received --> 0

Received --> 1

Received --> 0

Received --> 0

Received --> 0

Received --> 0

Received --> 1

Received --> 1

Received --> 1

Received --> 0

Received --> 0

Received --> 1

timeout

sending again--> 0

ack--> 1

sending --> 1

EXPERIMENT 9

9. Write a program for congestion control using leaky bucket algorithm

PROGRAM:

```
print("Enter bucket size, outgoing rate, number of inputs and
incoming size")

bucketsize = int(input())

outgoing = int(input())

n = int(input())

incoming = int(input())

store=0

while n!= 0:

      print("Incoming size is ", incoming)

      if incoming <= (bucketsize-store):

            store += incoming

            print("Bucket  buffer  size  is  ",store,"  out  of  ",
            bucketsize)

      else:

            print("Packet loss : ",(incoming-(bucketsize-store)))

            store=bucketsize

            print("Bucket  buffer  size  is  ",store,"  out  of  ",
            bucketsize)

      store -= outgoing;

      print("After outgoing: " ,store," packets left out of ",
      bucketsize ,"in buffer")

      n=n-1
```

OUTPUT:

```
Enter bucket size, outgoing rate, number of inputs and
incoming size

300

50

2

200

Incoming size is 200

Bucket buffer size is    200 out of 300

After outgoing: 150 packets left out of 300 in buffer

Incoming size is 200

Packet loss: 50

Bucket buffer size is    300 out of 300

After outgoing: 250 packets left out of 300 in buffer
```

EXPERIMENT 10

10. Write a program to implement Dijkstra's algorithm to compute the shortest path through a graph.

PROGRAM:

```
INF = 1000


#  Search  minimum  function
def search_min(length, se, n):
   global   v
   mi = 100
   for i in range(n):
      if se[i] == 0:
         if length[i] < mi:
            mi  =  length[i]
            v = i
   return v


se = [0] * 10
length = []
path = []
graph = []


n = int(input("Enter No of Vertices: "))
print("Enter the adjacency matrix: ")
```

```
for i in range(n):
    graph.append(list(map(int, input().split())))

s = int(input("Enter Source node: "))

# INITIALIZATION PART
for i in range(n):
    if graph[s][i] == 0:
        length.append(INF)
        path.append(0)
    else:
        length.append(graph[s][i])
        path.append(s)

se[s] = 1
length[s] = 0

# ITERATION PART
c = 1
while c:
    c = 0
    j = search_min(length, se, n)
    se[j] = 1
    for i in range(n):
        if se[i] != 1:
            if graph[i][j] != 0:
```

```
            if length[j] + graph[i][j] < length[i]:
                length[i] = length[j] + graph[i][j]
                path[i] = j

    for i in range(0, n):
        if se[i] == 0:
            c += 1

# OUTPUT
print("From (source vertex) To ", s)
print("\tPath\t\tLength\t\tShortest path")

for i in range(n):
    if i != s:
        print("\t\t%d\t\t%d" % (i, length[i]), end='\t')
        j = i
        while j != s:
            print("\t%d->%d" % (j, path[j]), end='\t')
            j = path[j]
        print()
```

OUTPUT:

```
Enter No of Vertexes: 4
enter the adjacency matrix:
0 6 0 1
6 0 2 4
0 2 0 1
1 4 1 0
```

```
Enter Source node: 0

From(sourcevertex) To 0


 Path     Length    Shortest path

   1         4       1->2         2->3      3->0

   2         2       2->3         3->0

   3         1       3->0
```

WEEK-1

Representation of a Text Document in Vector Space Model and Computing Similarity between two documents.

AIM:

To compute the similarity between two text documents using **TF-IDF vectorization** and **Cosine Similarity** in Python.

DESCRIPTION:

Cosine Similarity – Definition:

Cosine Similarity is a metric used to measure the similarity between two non-zero vectors by calculating the **cosine of the angle** between them.

In the context of text documents, it compares two documents represented as TF-IDF vectors and returns a value between **0** and **1**:

- **1** indicates identical documents (high similarity).
- **0** indicates completely different documents.

It is widely used in **text mining**, **information retrieval**, and **natural language processing** to compare the semantic similarity between documents.

Vector Space Model – Definition:

The Vector Space Model is a mathematical model used to represent text documents as vectors in a multi-dimensional space.

Each unique term in the entire corpus forms a separate dimension. The values in the vector are typically **term weights**, such as **TF-IDF scores**, which indicate the importance of a word in a document.

In this model:

- Documents are treated as vectors.
- Similarity can be computed using mathematical techniques like **cosine similarity**.

This model enables efficient **document comparison**, **ranking**, and **clustering** in text-based applications.

This experiment demonstrates how to measure the similarity between two documents using Natural Language Processing techniques.

The program uses **TF-IDF (Term Frequency–Inverse Document Frequency)** to convert textual data into numerical form, where each word is assigned a weight based on its importance in the document and across documents.

Once the documents are represented as TF-IDF vectors, the **cosine similarity** between them is calculated. Cosine similarity is a metric that quantifies the cosine of the angle between two vectors in a multi-dimensional space. A similarity score close to 1 indicates high similarity, while a score near 0 indicates low similarity.

Cosine Similarity – Formula:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Where:

- A and B are two vectors (e.g., TF-IDF vectors of two documents).
- $A \cdot B$ is the **dot product** of the vectors.
- $\|A\|$ and $\|B\|$ are the **magnitudes (lengths)** of vectors A and B.

Step-by-Step Process to Calculate Cosine Similarity:

1. **Convert documents into vectors**
   Use a technique like **TF-IDF** to convert each document into a numeric vector.
2. **Compute the dot product $A \cdot B$**
   Multiply corresponding values of the two vectors and sum them.
3. **Compute the magnitude of each vector**

$$\|A\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

$$\|B\| = \sqrt{b_1^2 + b_2^2 + \cdots + b_n^2}$$

4. **Apply the cosine similarity formula**

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Interpretation of Result:

- **1** → Documents are exactly the same in direction (very similar).
- **0** → Documents are completely different (orthogonal vectors).
- **Closer to 1** → More similar.

PROGRAM:

```
doc1 = "Machine learning is amazing and fun"
doc2 = "Deep learning and machine learning are parts of artificial intelligence" documents = [doc1, doc2]
from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine_similarity
# Create TF-IDF vectorizer vectorizer = TfidfVectorizer()
# Fit and transform the documents
tfidf_matrix = vectorizer.fit_transform(documents)
# Display the feature names (terms)
print("Vocabulary:", vectorizer.get_feature_names_out())
# Display TF-IDF matrix
print("TF-IDF Matrix:\n", tfidf_matrix.toarray())
# Compute cosine similarity between the two documents similarity =
cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])
print("Cosine Similarity between doc1 and doc2:", similarity[0][0])
```

OUTPUT:

```
Vocabulary: ['amazing' 'and' 'are' 'artificial' 'deep' 'fun' 'intelligence' 'is'
 'learning' 'machine' 'of' 'parts']
TF-IDF Matrix:
[[0.447 0.447 0.    0.    0.    0.447 0.    0.447 0.447 0.447 0.    0.    ]
 [0.    0.265 0.265 0.265 0.265 0.    0.265 0.    0.529 0.529 0.265 0.265]]
Cosine Similarity between doc1 and doc2: 0.487
```

WEEK-2

Pre-processing of a Text Document: stop word removal and stemming.

AIM:

To preprocess a text document by performing stop word removal and stemming

DESCRIPTION:

Understanding Stop Words

Stop words in Natural Language Processing (NLP) refer to the most common words in a language. Examples include "and", "the", "is", and others that do not provide significant meaning and are often removed to speed up processing without losing crucial information. For this purpose, Python's Natural Language Tool Kit (NLTK) provides a pre-defined list of stop words.

Introduction to Stemming

Stemming is a technique that reduces a word to its root form. Although the stemmed word may not always be a real or grammatically correct word in English, it does help to consolidate different forms of the same word to a common base form, reducing the complexity of text data. This simplification leads to quicker computation and potentially better performance when implementing Natural Language Processing (NLP) algorithms, as there are fewer unique words to consider.

For example, the words "run", "runs", "running" might all be stemmed to the common root "run". This helps our algorithm understand that these words are related and they carry a similar semantic meaning.

PROGRAM:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize from
nltk.stem import PorterStemmer nltk.download('punkt')
nltk.download('stopwords')
```

```
def preprocess_text(text):
    text = text.lower()
    words = word_tokenize(text)

    stop_words = set(stopwords.words('english'))

    filtered_words = [word for word in words if word.isalnum() and word not in stop_words]

    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in filtered_words]
    return stemmed_words
text = "Machine learning algorithms are revolutionizing the world of artificial intelligence."
print("Orginal Text:",text)
processed = preprocess_text(text)
processed_text = ' '.join(processed)
print("Processed Text:", processed_text)
print("Preprocessed Words:", processed)
```

OUTPUT:

Orginal Text: Machine learning algorithms are revolutionizing the world of artificial intelligence.

Processed Text: machin learn algorithm revolution world artifici intellig

Preprocessed Words: ['machin', 'learn', 'algorithm', 'revolution', 'world', 'artifici', 'intellig']

WEEK-14

Construction of an Inverted Index for a given document collection comprising of at least 50 documents with a total vocabulary size of at least 1000 words.

AIM:

To construct an inverted index for a collection of 50 documents and vocabulary of 1000 words.

DESCRIPTION:

An **inverted index** is a fundamental data structure in the field of information retrieval. It is designed to enable fast and efficient full-text searches over a large collection of documents. Instead of storing documents sequentially as they are, an inverted index reverses the natural relationship between documents and terms by mapping each **unique word (term)** in the corpus to a list of **documents** in which it appears. This allows for quick lookup of all documents containing a given word. In constructing an inverted index, the document collection must undergo several preprocessing steps to normalize and clean the data. These steps typically include

- converting all text to lowercase, removing punctuation and stopwords (common, non-informative words like “the,” “is,” “and”),
- tokenizing the text into words,
- applying stemming or lemmatization to reduce words to their root forms (e.g., “running” becomes “run”).

PYTHON CODE:

```
import os import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer

from collections import defaultdict import json

nltk.download('punkt')

 nltk.download('stopwords')
```

```
def preprocess(text):
    text = text.lower()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english')) stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in tokens if word.isalnum() and word not in
    stop_words]
    return words documents = {}
for filename in os.listdir():
    if filename.endswith(".txt"):
        with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
            text = f.read()
            documents[filename] = preprocess(text)
print(f"Total documents loaded: {len(documents)}")
inverted_index = defaultdict(set)
for doc_id, words in documents.items():
    for word in set(words): # avoid duplicates per document
        inverted_index[word].add(doc_id)
vocab_size = len(inverted_index)
print(f"\nVocabulary Size: {vocab_size} words")
print("\nSample inverted index terms:")
for term in list(inverted_index)[:10]:
    print(f"{term}: {sorted(inverted_index[term])}")
```

OUTPUT:

```
Total documents loaded: 11
defaultdict(<class 'set'>, {'today': {'HI.txt'}, 'work': {'HI.txt'}, 'warm': {'untitled4.txt', 'HI.txt'},
'hi': {'HI.txt'}, 'professor': {'HI.txt'}, 'aditya': {'HI.txt'}, 'sushuma': {'HI.txt'}, 'assist': {'HI.txt'},
```

'sunni': {'untitled4.txt'}})

Vocabulary Size: 9 words

Sample inverted index terms:

hi: ['HI.txt']

sushuma: ['HI.txt']

today: ['HI.txt']

aditya: ['HI.txt']

work: ['HI.txt']

professor: ['HI.txt']

assist: ['HI.txt']

warm: ['HI.txt', 'untitled4.txt']

sunni: ['untitled4.txt']

WEEK-4

Classification of a set of Text Documents into known classes (You may use any of the Classification algorithms like Naive Bayes, Max Entropy, Rochio"s, Support Vector Machine). Standard Datasets will have to be used to show the results.

AIM:

To perform classification of text documents into known classes.

DESCRIPTION:

Introduction to Text Classification

Text classification is a fundamental task in Natural Language Processing (NLP) where a piece of text (e.g., a document, sentence, or paragraph) is assigned to one or more predefined categories. Applications include:

- Spam detection
- Sentiment analysis
- Topic labelling
- News categorization

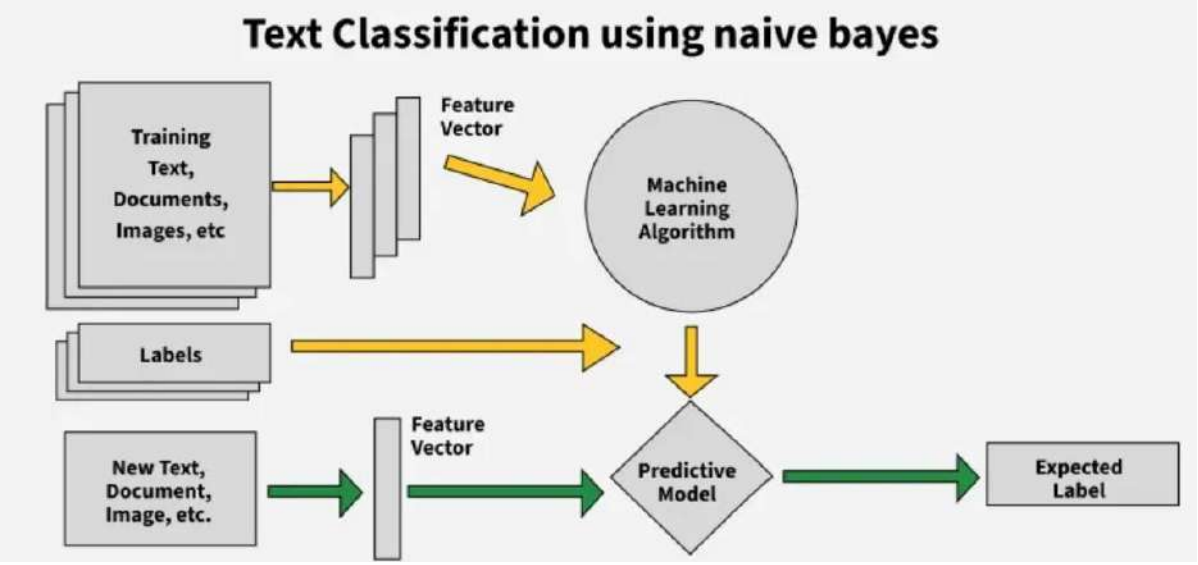
Naive Bayes

Naive Bayes is a classification algorithm that uses probability to predict which category a data point belongs to, assuming that all features are unrelated. It is named as "Naive" because it assumes the presence of one feature does not affect other features.

Why Naive Bayes?

The Naive Bayes classifier is a popular probabilistic machine learning algorithm based on Bayes' Theorem with an assumption of independence among features. It is particularly well- suited for text classification due to:

- **Efficiency** in handling high-dimensional data (like text)
- **Robustness** even with small training datasets
- **Scalability** and speed of training and prediction



PROGRAM:

```
from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report

categories = ['sci.space', 'rec.sport.hockey', 'comp.graphics', 'alt.atheism']

newsgroups= fetch_20newsgroups(subset='all',categories=categories,shuffle=True, random_state=42)

print(f"Total documents: {len(newsgroups.data)}")

print(f"Target classes: {newsgroups.target_names}")

X_train, X_test, y_train, y_test = train_test_split (newsgroups.data, newsgroups.target, test_size=0.2,
random_state=42 )

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)

X_test_tfidf = vectorizer.transform(X_test)

nb = MultinomialNB()
```

```
nb.fit(X_train_tfidf, y_train)
y_pred = nb.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, target_names=newsgroups.target_names))
for i in range(5):
    print("\nText:\n", X_test[i])
    print("Actual:", newsgroups.target_names[y_test[i]])
    print("Predicted:", newsgroups.target_names[y_pred[i]])
```

OUTPUT:

Accuracy: 0.9840425531914894

Classification Report:

	precision	recall	f1-score	support
alt.atheism	1.00	1.00	1.00	152
comp.graphics	0.96	0.99	0.97	196
rec.sport.hockey	0.99	1.00	1.00	194
sci.space	0.99	0.95	0.97	210
accuracy			0.98	752
macro avg	0.99	0.99	0.99	752
weighted avg	0.98	0.98	0.98	752

WEEK-5

Text Document Clustering using K-means. Demonstrate with a standard dataset and compute performance measures- Purity, Precision, Recall and F-measure.

AIM:

To cluster a document using K-Means and compute the performance measures.

DESCRIPTION:

Text Clustering

Text document clustering is an **unsupervised machine learning** technique used to **group similar documents** into clusters based on their content. Unlike classification, clustering doesn't use labeled data. It's widely used in:

- Document organization
- Topic discovery
- Information retrieval

K-Means Clustering Algorithm

K-means clustering is a type of unsupervised learning method, which is used when we don't have labelled data as in our case, we have unlabelled data (means, without defined categories or groups). The goal of this algorithm is to find groups in the data, whereas the no. of groups is represented by the variable K. The data have been clustered on the basis of high similarity points together and low similarity points in the separate clusters.

Algorithm Steps:

1. **Preprocessing**: Convert raw documents into a numerical format (e.g., TF-IDF vectors).
2. **Initialization**: Choose k initial centroids randomly.
3. **Assignment Step**: Assign each document to the nearest centroid (based on Euclidean or Cosine distance).
4. **Update Step**: Recompute centroids based on the assigned documents.

5. **Repeat** the assignment and update steps until convergence (no change in centroids or assignments).

3. Text Representation using TF-IDF

Before clustering, text documents are converted into vector form using:

- **Bag of Words (BoW)** or
- **TF-IDF (Term Frequency-Inverse Document Frequency)**

TF-IDF reduces the impact of commonly used words and emphasizes more informative terms.

Performance Evaluation Metrics

In clustering (especially with known ground truth), we compare the clustering results with actual labels using the following metrics:

1. Purity

Purity measures the extent to which clusters contain documents from primarily one class.

Formula:

$$\text{Purity} = \frac{1}{N} \sum_k \max_j |C_k \cap L_j|$$

- Ck: Cluster k
- Lj : Class j
- N: Total number of documents

A higher purity means better clustering. Maximum value = 1.

2. Precision (per class)

Precision measures how many documents assigned to a cluster are actually relevant (i.e., from the correct class).

Formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

TP: True Positives

FP: False Positives

3. Recall (per class)

Recall measures how many relevant documents (from a true class) are correctly assigned to a cluster.

Formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

TP: True Positives

FN: False Negatives

4. F-measure (F1 Score)

F-measure is the harmonic mean of precision and recall. It balances both metrics.

Formula:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

PROGRAM:

```
import numpy as np

from sklearn.datasets import fetch_20newsgroups

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.metrics import precision_score, recall_score, f1_score

from scipy.stats import mode
```

```
# Step 2: Load Dataset (using 4 categories for clarity)
categories = ['alt.atheism', 'comp.graphics', 'sci.space', 'talk.religion.misc']
newsgroups = fetch_20newsgroups(subset='all', categories=categories, remove=('headers', 'footers',
'quotes'))
# Step 3: Convert Text to TF-IDF Features
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
X = vectorizer.fit_transform(newsgroups.data)
y_true = newsgroups.target
# Step 4: Apply K-Means Clustering
k = len(categories)
kmeans = KMeans(n_clusters=k, random_state=42)
y_pred = kmeans.fit_predict(X)


def purity_score(y_true, y_pred):
    clusters = np.unique(y_pred)
    classes = np.unique(y_true)
    contingency_matrix = np.zeros((len(clusters), len(classes)))
    for i, cluster in enumerate(clusters):
        indices = np.where(y_pred == cluster)[0]
        true_labels = y_true[indices]
        if len(true_labels) == 0:
            continue
        most_common = mode(true_labels, keepdims=True).mode[0]
        count = np.sum(true_labels == most_common)
        j = np.where(classes == most_common)[0][0]
```

```
            contingency_matrix[i][j] = count
return np.sum(np.max(contingency_matrix, axis=1)) / np.sum(contingency_matrix)
# Create a label mapping from cluster to majority class
def map_clusters_to_labels(y_true, y_pred):
    label_mapping = {}
    for cluster in np.unique(y_pred):
        indices = np.where(y_pred == cluster)[0]
        if len(indices) == 0:
            continue
        majority_label = mode(y_true[indices], keepdims=True).mode[0]
        label_mapping[cluster] = majority_label
    # Map each prediction to the true label
    mapped_preds = np.array([label_mapping[cluster] for cluster in y_pred])
    return mapped_preds
y_pred_mapped = map_clusters_to_labels(y_true, y_pred)
# Compute Metrics
purity = purity_score(y_true, y_pred)
precision = precision_score(y_true, y_pred_mapped, average='macro') recall =
recall_score(y_true, y_pred_mapped, average='macro')
f1 = f1_score(y_true, y_pred_mapped, average='macro')
# Print Results
print("Purity Score:", round(purity, 4))
print("Precision:", round(precision, 4))
print("Recall:", round(recall, 4))
print("F1-Score:", round(f1, 4))
```

OUTPUT:

Purity Score: 1.0

Precision: 0.6767

Recall: 0.5225

F1-Score: 0.5253

WEEK-6

Crawling/ Searching the Web to collect news stories on a specific topic (based on user input). The program should have an option to limit the crawling to certain selected websites only.

AIM:

To Crawl/ Search the Web to collect news stories on a specific topic.

DESCRIPTION:

Crawling

Crawling is the process by which a program (called a **crawler** or **spider**) automatically browses the web to collect and download information from websites.

Introduction

Web crawling is the automated process of systematically browsing and extracting information from web pages. In the context of news story collection, a crawler is programmed to search the internet for news articles related to a **specific topic** provided by the user. The goal is to collect relevant content from the web and present it in a usable format.

How It Works

1. **User Input**
   - The user specifies a **topic** (e.g., "Climate Change", "Artificial Intelligence", "Olympics 2024").
   - Optionally, the user can **limit the search to specific websites** (e.g., *bbc.com*, *nytimes.com*).
2. **URL Queue Initialization**
   - The crawler starts with a set of **seed URLs** (either from the user or pre-defined news portals).
3. **Fetching Pages**
   - The crawler downloads the HTML content of these web pages.

4. **Parsing & Extraction**
   - The HTML is analyzed to extract:
     - Headlines
     - Publication dates
     - Article content
     - Author names
   - Relevant **links to other news pages** are also extracted.
5. **Filtering by Topic**
   - Using **keyword matching** or **Natural Language Processing (NLP)**, the crawler selects only those articles related to the given topic.
6. **Storing Results**
   - The collected news stories are saved in a database or file for further reading, analysis, or summarization.

Install package: pip install serpapi beautifulsoup4 requests About package:

- **serpapi**: A Python wrapper for SerpAPI, which lets you retrieve Google/Bing/News results through their API (requires a SerpAPI key).
- **beautifulsoup4**: A Python library for parsing HTML and XML documents; commonly used to extract data from web pages.
- **requests**: A popular library to send HTTP requests easily in Python.

PROGRAM:

```
pip install serpapi beautifulsoup4 requests

import requests

from bs4 import BeautifulSoup

topic = input("Enter the news topic to search for: ")
```

```
websites = input("Enter comma-separated websites to limit crawling (e.g., bbc.com,cnn.com):
                ").split(',')
SERP_API_KEY = '8d6bc2b3eef2e66c277a5a34be29b70d490834e929934539b15ae91c71dd569c'
search_url = 'https://serpapi.com/search.json'
def search_news(topic, websites):
        all_results = []
        for site in websites:
                params = {
                        "engine": "google",
                        "q": f"{topic} site:{site.strip()}",
                        "api_key": SERP_API_KEY
                }
                response = requests.get(search_url, params=params)
                data = response.json()
                if "organic_results" in data:
                        for result in data["organic_results"]:
                                title = result.get("title")
                                link = result.get("link")
                                snippet = result.get("snippet", "")
                                all_results.append((title, link, snippet))
        return all_results
def display_results(results):
        for idx, (title, link, snippet) in enumerate(results, start=1):
                print(f"\nNews {idx}:")
                print(f"Title  : {title}")
```

```
            print(f"URL    : {link}")
            print(f"Summary : {snippet}")
results = search_news(topic, websites)
if results:
    display_results(results)
else:
    print("No results found.")
```

OUTPUT:

Enter the news topic to search for: AI in healthcare

Enter comma-separated websites to limit crawling (e.g., bbc.com,cnn.com): bbc.com,cnn.com

News 1:

Title : AI in healthcare: what are the risks for the NHS?

URL : https://www.bbc.com/news/articles/c6233x9k4dlo

Summary : Generative AI will be transformative for NHS patient outcomes, a senior government advisor says.

News 2:

Title : How AI can spot diseases that doctors aren't looking for

URL : https://www.bbc.com/news/articles/c9q7zqy1xlpo

Summary : AI can take a second look at medical scans and flag up potential problems that doctors might not see.

News 3:

Title : How AI Has Transformed Healthcare's Future

URL : https://www.bbc.com/storyworks/hpe-greenlake/how-ai-has-transformed-healthcares-future

Summary : AI can link seemingly unrelated information to reveal new research pathways that yield better results. For example, AI models have identified potential ...

News 4:

Title : Hospitals will use AI to speed up patient care

URL : https://www.bbc.com/news/articles/cye0yywdegdo

Summary : Hospitals across the region are to use artificial intelligence (AI) technology to reduce unnecessary admissions and lengthy stays, ...

News 5:

Title : Can AI help modernise Ireland's healthcare system?

URL : https://www.bbc.com/news/articles/cly7yxm3py5o

Summary : Ireland is investing billions of euros to revamp its healthcare service - will AI help?

News 6:

Title : How artificial intelligence is matching drugs to patients

URL : https://www.bbc.com/news/business-65260592

Summary : Health-tech firms around the world are increasingly using AI to help tailor drugs for patients.

WEEK-7

To parse XML text, generate Web graph and compute topic specific page rank Parse XML Text

AIM:

To parse XML text, generate Web graph and compute topic specific page rank .

DESCRIPTION:

Parsing XML means reading the XML data, understanding its structure, and extracting the relevant information from it.

- Purpose: Often, XML files store web crawl data — URLs, hyperlinks, anchor texts, and metadata.
- Process:
    1. Load XML file into memory.
    2. Use an XML parser (like Python's xml.etree.ElementTree or lxml) to navigate the tree structure.
    3. Extract required fields — e.g., <page>, <link>, <title>, <content>.
    4. Store these in data structures like dictionaries or adjacency lists for further graph processing.

Generate Web Graph

The web graph is a directed graph where:

- Nodes (vertices) = web pages
- Edges = hyperlinks from one page to another
- Construction:
    1. Create a mapping from URL → node ID.
    2. For each <page> in the XML, list its outgoing <link>s.
    3. Build an adjacency list or adjacency matrix.
- Purpose: This graph becomes the foundation for PageRank calculations.

Compute Topic-Specific PageRank

Topic-specific PageRank modifies the standard PageRank by biasing the random surfer towards a set of topic-related pages.

- Idea: Instead of teleporting uniformly to any page, the algorithm teleports more often to pages relevant to a given topic (e.g., “sports” or “technology”).

- Steps:
    1. Identify a topic seed set — pages known to be about the topic.
    2. Initialize teleportation vector v where:
        - Higher values for topic-relevant pages
        - Lower (or zero) for irrelevant pages
    3. Use the PageRank equation:

        $$PR=\alpha MPR+(1-\alpha)v$$

        - M: transition matrix from the web graph
        - α (alpha): damping factor (usually 0.85)
        - v: topic-biased teleportation vector
    4. Iterate until convergence (small changes between iterations).

PROGRAM:

```
import xml.etree.ElementTree as ET

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

# Step 1: Parse XML

def parse_xml(xml_text):

    tree = ET.fromstring(xml_text)

    graph = {}
```

```
    topics_map = {}
    for page in tree.findall('page'):
        title = page.find('title').text.strip()
        links = [link.text.strip() for link in page.findall('link')]
        topics = page.find('topics').text.strip().split(",") if page.find('topics') is not
                    None else []
        graph[title] = links
        topics_map[title] = [t.strip() for t in topics]
    return graph, topics_map
```

 # Step 2: Build Adjacency Matrix

```
def build_adj_matrix(graph):
    pages = list(graph.keys())
    idx = {page: i for i, page in enumerate(pages)}
    n = len(pages)
    M = np.zeros((n, n))
    for page, links in graph.items():
        if links:
            for link in links:
                if link in idx:
                    M[idx[link]][idx[page]] = 1 / len(links)
        else:
            M[:, idx[page]] = 1 / n # dangling node handling
    return M, pages
```

 # Step 3: Compute Topic-Specific PageRank

```
def topic_specific_pagerank(M, pages, topics_map, topic, d=0.85, tol=1e-6, max_iter=100):
```

```
    n = len(pages)
    teleport = np.array([1.0 if topic in topics_map[p] else 0.0 for p in pages])
    if teleport.sum() == 0:
        teleport = np.ones(n)
    teleport = teleport / teleport.sum() # normalize
    r = np.ones(n) / n # initial rank
    for i in range(max_iter):
        r_new = d * M @ r + (1 - d) * teleport
        if np.linalg.norm(r_new - r, 1) < tol:
            break
        r = r_new
    return dict(zip(pages, r))
```

 # Step 4: Visualize the Web Graph with Topic Highlight

```
def draw_web_graph(graph, topics_map, topic):
    G = nx.DiGraph()
    for page, links in graph.items():
        for link in links:
            G.add_edge(page, link)
```

 # Node colors: highlight pages having the topic

```
node_colors = []
for page in G.nodes():
    if topic in topics_map.get(page, []):
        node_colors.append("lightgreen") # highlight topic pages
    else:
```

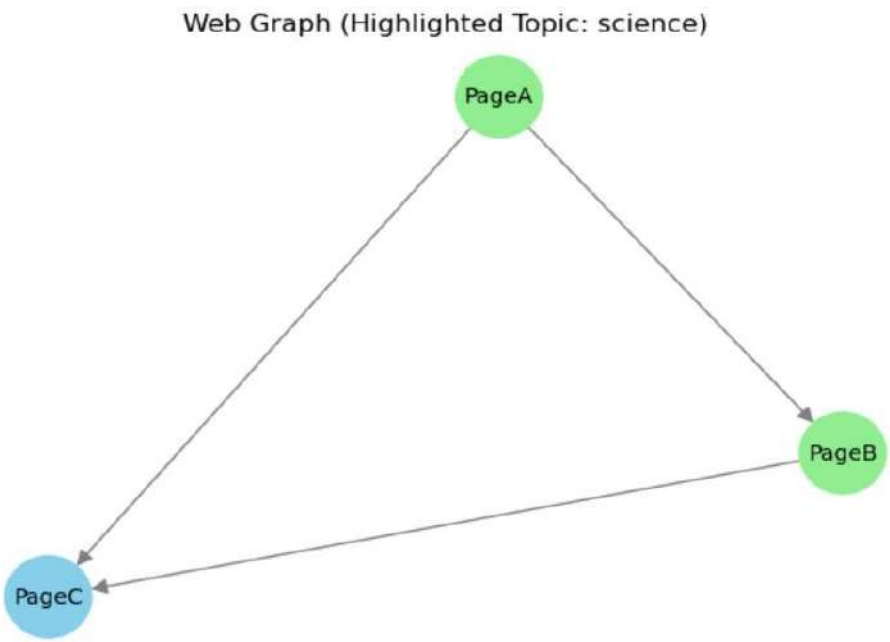
```
                node_colors.append("skyblue")        # normal pages
plt.figure(figsize=(6, 4))
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_color=node_colors, node_size=1500,
                font_size=10, arrowsize=15, edge_color="gray")
plt.title(f"Web Graph (Highlighted Topic: {topic})")
plt.show()
```

 # Step 5: Input and Execute

```
xml_text = '''<web>
    <page>
        <title>PageA</title>
        <link>PageB</link>
        <link>PageC</link>
        <topics>science,education</topics>
    </page>
    <page>
        <title>PageB</title>
        <link>PageC</link>
        <topics>science</topics>
    </page>
    <page>
        <title>PageC</title>
            <topics>sports</topics>
    </page>
</web>'''
```

```
graph, topics_map = parse_xml(xml_text)

M, pages = build_adj_matrix(graph)

# Draw the web graph with topic highlighting

topic = "science"

draw_web_graph(graph, topics_map, topic)

# Compute topic-specific PageRank

ranks = topic_specific_pagerank(M, pages, topics_map, topic)

print("\nTopic-Specific PageRank (Topic: science):")

for page, score in sorted(ranks.items(), key=lambda x: -x[1]):

        print(f"{page}: {score:.4f}")
```

OUTPUT:




```
Topic-Specific PageRank (Topic: science):
PageC: 0.4849
PageB: 0.3027
PageA: 0.2124
```

WEEK-8

Implement Matrix Decomposition and LSI for a standard dataset.

AIM:

To implement matrix decomposition and LSI for a standard dataset.

DESCRIPTION:

Matrix Decomposition

Matrix decomposition (or matrix factorization) is a mathematical technique used to break down a large matrix into smaller, simpler matrices.

- In text mining, we typically start with a document-term matrix (DTM) or TF-IDF matrix where:
    - Rows = documents
    - Columns = terms (words)
    - Values = frequency or importance of terms in documents

Latent Semantic Indexing (LSI)

Latent Semantic Indexing (also called Latent Semantic Analysis, LSA) is a technique in information retrieval and natural language processing that uses SVD on the term-document matrix.

Idea behind LSI

- Natural language has synonyms (different words with similar meaning) and polysemy (same word with multiple meanings).
- A raw term-document matrix treats each word independently, which misses semantic relationships.
- LSI reduces the matrix dimensions to capture hidden (latent) semantic structures in text.

Process of LSI

1. Construct a TF-IDF matrix from the dataset.
2. Apply Truncated SVD to decompose into lower-rank matrices.
3. Represent documents and terms in this reduced semantic space.
4. Use cosine similarity or other metrics to find similarity between documents/queries**. Singular Value Decomposition (SVD)**

- SVD is a mathematical technique that breaks a large matrix into three smaller matrices.
- For a document-term matrix (like TF-IDF), SVD helps find patterns/relationships between terms and documents.
- It identifies the most important concepts (latent topics) in the data

PROGRAM:

```
from sklearn.datasets import fetch_20newsgroups

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

# Step 1: Load dataset (subset for speed)

categories = ['sci.space', 'rec.sport.hockey', 'comp.graphics']

newsgroups = fetch_20newsgroups(subset='train', categories=categories, remove=('headers',

                    'footers', 'quotes'))

# Step 2: TF-IDF Vectorization

vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)

X_tfidf = vectorizer.fit_transform(newsgroups.data)

print(f"Original TF-IDF shape: {X_tfidf.shape}") # (docs x terms)

# Step 3: SVD for LSI (Latent Semantic Indexing)

k = 100 # number of latent dimensions

svd = TruncatedSVD(n_components=k)

X_lsi = svd.fit_transform(X_tfidf)

print(f"Reduced LSI shape: {X_lsi.shape}") # (docs x k topics)

# Step 4: Show similarity between some documents

def show_similar_docs(query_idx, top_n=5):
```

```
    similarities = cosine_similarity([X_lsi[query_idx]], X_lsi)[0]
    top_indices = similarities.argsort()[::-1][1:top_n+1]
    print(f"\nQuery Document {query_idx}:\n{newsgroups.data[query_idx][:300]}...\n")
    print("Top similar documents:")
    for i in top_indices:
        print(f"\nDoc #{i} (Similarity: {similarities[i]:.3f}):\n{newsgroups.data[i][:300]}...")
```

 # Example: Show top 5 similar documents to doc #0

```
show_similar_docs(query_idx=0, top_n=5)
```

OUTPUT:

```
Original TF-IDF shape: (1777, 1000)
Reduced LSI shape: (1777, 100)
Query Document #0:
Mike Vernon is now 3 wins 11 losses plus that All-Star game debacle in
afternoon games during his career...with another afternoon game with
Los Angeles next Sunday...has the ABC deal doomed the Flames?...
Top similar documents:
Doc #342 (Similarity: 0.686):
Dale Hunter ties the game, scoring his third goal of the game
with 2.7 seconds remaining in regulation.
    You could feel it coming on.
    "Due to contractual agreements, ESPN will be unable to carry
the rest of this game live, so that we may show you a worthless early-season battle between th...
Doc #1208 (Similarity: 0.658):
Showing a meaningless (relatively) baseball game over the overtime of
game that was tied up with less than 3 seconds left on the clock?
Gimme a break!  Where does ESPN get these BRILLIANT decisions from?...
```

WEEK-9

Mining Twitter to identify tweets for a specific period (and/or from a geographical location) and identify trends and named entities.

AIM:

To mine twitter for identifying tweets for a specific period.

DESCRIPTION

This program is designed to mine Twitter data in order to analyze social media activity within a specified **time period** and/or **geographical location**. By leveraging the Twitter API, it collects tweets that match a given query or hashtag, applies filters such as date ranges and geotags, and then processes the data to extract meaningful insights.

The workflow of the program includes:

1. **Data Collection**

- Connects to the Twitter API using a bearer token.
- Retrieves tweets based on user-defined criteria such as keywords, hashtags, time period, and geographical coordinates.

2. **Preprocessing**

- Cleans tweet text by removing URLs, mentions, hashtags, emojis, and stopwords.
- Normalizes the text for further linguistic analysis.

3. **Trend Analysis**

- Identifies the most frequent hashtags, keywords, and topics used during the selected period or within the chosen location.
- Detects trending discussions by analyzing tweet frequency patterns.

4. **Named Entity Recognition (NER)**

- Applies natural language processing (NLP) techniques to detect and classify named entities such as persons, organizations, locations, products, or events.
- Helps to understand which entities dominate public discourse in the given timeframe or region.

5. **Visualization & Insights**

- Generates word clouds, frequency charts, and time-based trend graphs.

- Provides insights into emerging topics, popular hashtags, and influential entities.

PROGRAM:

```
import tweepy

# Replace with your own Bearer Token from Twitter Developer Portal

bearer_token =
"AAAAAAAAAAAAAAAAAAAAAADhM4QEAAAAAEhGXBfqa4kNwgb3%2F3XEC8JceL
Ys%3D0AVX5bRfhoQTvuRjjokbg7zOQ6egn1VOGtL2xEXIW4N7IGsX9P"

# Initialize Tweepy client with bearer token

client = tweepy.Client(bearer_token=bearer_token)

# Define your search query

query = "AI OR Machine Learning"

# Fetch recent tweets matching the query

tweets = client.search_recent_tweets(

    query=query,

    max_results=100,            # maximum results per request (up to 100)

    tweet_fields=['created_at', 'text']  # request tweet creation time and text

)

# Check if tweets are returned

if tweets.data is not None:

    # Print tweet creation date and text

    for tweet in tweets.data:

        print(f"Created at: {tweet.created_at}")

        print(f"Tweet text: {tweet.text}\n")

else:

    print("No tweets found for this query.")
```

OUTPUT:

Created at: 2025-09-23 03:51:48+00:00

Tweet text: RT @leiane1: Good morning, family

How are you?

The @recallnet Arena is NOW open.

Trade proven, high-volume pairs with real liquidity.…

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @icanvardar @stripe Stripe is becoming an ai labs

Created at: 2025-09-23 03:51:48+00:00

Tweet text: RT @GaiAIio: 🚀 GaiAI Discord is live!

Join our growing community of creators, developers, and Web3 AI explorers.

Discuss ideas, share gen…

Created at: 2025-09-23 03:51:48+00:00

Tweet text: RT @psicolut: a virginia sambando daquele jeito como rainha de bateria e voce aí se cobrando pra tirar um projeto do papel porque ainda não…

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @JnglJourney LOL....AI...UFOs....the spooky ghouls of Halloween arriving early....

Or is it EU countries confabulating fake narratives to blame Russia for these mystery sightings.

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @OpenledgerFdn @kbwofficial @OpenledgerHQ Openledger is really building the fair layer of the OPEN internet AI

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @69on_ai 这是哪部作品的人物呀

Created at: 2025-09-23 03:51:48+00:00

Tweet text: RT @FractionAI_xyz: Here’s a crazy thought:

Every Tuesday, we’ve been shipping something new and exciting, week in and week out.

This sh…

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @darwinmda_ @jaofranko As vezes é mais sobre o traços do artista, mas acho que se ele fizesse o cara chorando ai sim seria

Created at: 2025-09-23 03:51:48+00:00

Tweet text: RT @skywongraveee: ผมเก็บสิ่งนี้มาได้มันคืออะไร ถาม ai ก็ไม่รู้ ใครทราบรบกวนแจ้งที

#MuTeLuvNotMyFatherEP1 https://t.co/VzYjCk7Dv5

Created at: 2025-09-23 03:51:48+00:00

Tweet text: @KAMADAN_AI めっちゃ共感です！

WEEK-10

Implementation of PageRank on Scholarly Citation Network.

AIM:

To implement Page Rank on scholarity Citation Network.

DESCRIPTION:

What is PageRank?

- PageRank is an algorithm originally developed by Google founders Larry Page and Sergey Brin.
- It measures the *importance* of nodes in a directed graph, based on the idea that a node (like a web page or research paper) is important if other important nodes link (or cite) it.
- In a scholarly citation network:
  - **Nodes = Research papers**
  - **Edges = Citations (directed links from citing paper to cited paper) Use of**

PageRank in citation networks

- To identify **influential research papers**.
- A paper is important not just because it has many citations, but because it is cited by other important papers.
- It helps rank scholarly articles in terms of *impact* rather than just *counting citations.*

Example

Suppose we have a small citation network:

- Paper1 → cites Paper2, Paper3
- Paper2 → cites Paper3
- Paper3 → cites Paper1

This forms a cycle. Running PageRank will eventually distribute scores showing which paper is most central in the citation loop.

PROGRAM:

```
pip install --upgrade numpy scipy network
import networkx as nx
# Example scholarly citation network
# Each node is a paper, edges represent citations

citations = {
    "Paper1": ["Paper2", "Paper3"],
    "Paper2": ["Paper3"],
    "Paper3": ["Paper1"],
    "Paper4": ["Paper2", "Paper3"],
    "Paper5": ["Paper3", "Paper4"]
}

# Build directed graph G = nx.DiGraph()
for paper, cited_papers in citations.items():
    for cited in cited_papers:
        G.add_edge(paper, cited)
# Compute PageRank manually (no scipy backend needed) pagerank_scores = nx.pagerank(G,
alpha=0.85, max_iter=100) print("\n PageRank Scores:")
for paper, score in pagerank_scores.items():
    print(f"{paper}: {score:.4f}")
```

OUTPUT:

```
PageRank Scores:
Paper1: 0.3515
Paper2: 0.1975
Paper3: 0.3782
Paper4: 0.0428
Paper5: 0.0300
```