



















Date :

```
// void type - for functions that return nothing

function greet(): void {
  console.log("Hello from a void function!");
}

greet();

// never type - for functions that never return

function throwError(message: string): never {
  throw new Error(message);
}

// throwError("Something went wrong!"); // Uncomment to test
```

### Steps to Run:

1. Save your code in a file like `types.ts`
2. Compile it: **`tsc types.ts`**  
It will generate a `types.js` file.
3. Run it with Node.js: **`node types.js`**

**OUTPUT:**

Name: Alice

Age: 30

Is Student: true

Any type value: 100

Unknown as string: I MIGHT BE ANYTHING

Null value: null

Undefined value: undefined

Hello from a void function!













Date :

### Steps to Run:

1. Save your code in a file like `2d.ts`
2. Compile it: **`tsc 2d.ts`**  
It will generate a `types.js` file.
3. Run it with Node.js: **`node 2d.js`**

**OUTPUT:**

Hello, my name is Alice.

I am 30 years old.

Gender: Female

Hello, my name is Bob.

I am 20 years old.

Gender: Male

Student ID: 101

Gender: Male





**f. Write a program to understand generics with variables, functions, and constraints.**

## What are Generics?

## Generics in TypeScript — a powerful feature that allows you to write reusable, type-safe code.

**Generics** enable writing code that works with **any data type**, while still preserving **type safety**.

◆ **Syntax:**

```
function identity<T>(arg: T): T {
  return arg;
}
```



### 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.





Date :

**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

Date :

## 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
2. Debug backend issues
3. No code required
4. Save & share requests
5. Automate testing

You can send GET, POST, PUT, DELETE, etc., requests to any API and see the response instantly.

Helps find and fix issues in API response, parameters, headers, etc.

You don't need to write a program—just fill in the API URL, method, headers, and body.

You can save your request collections and share with team members.

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;
```

















### ☑ 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

**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.

























































## 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

Count: 0

Increment





### Step -by -step Procedure:

## 1. Folder Setup

my-app/

 $\vdash \text{src/}$ |  $\vdash$  App.js|  $\vdash$  App.css|  $\vdash$  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;
```





























Date :

```

    </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;
```



## 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 each path.
  - 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>
      </div>
    </Router>
  );
}
```















## 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**



- 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

## Hosts your database in the cloud

## Handles server maintenance

## Can run MongoDB without installing locally

## Cloud service with replication & backup

## MongoDB Compass

Lets you view & manage a database

## GUI tool for human interaction

Can run MongoDB without installing locally Requires a MongoDB server to connect to

## 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**.





Date :

- { "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')`

 $\{ \text{ok: 1} \}$ **Command:** `db.createCollection('class9')`
$$\{ \text{ok: 1} \}$$

### 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')
}
```

Date :

## 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  $\rightarrow$  criteria/conditions to match documents (like WHERE in SQL).

```
// Find all documents
```

**Command:** `db.class10.find()`

```
→
{
  _id: ObjectId('68b2b3c0f1189b6ebcbcd4420'),
  name: 'Raju',
  rollno: 1
}
{
  _id: ObjectId('68b2b48ef1189b6ebcbcd4421'),
  name: 'ramu',
  rollno: 2,
  phone: 123456789
}
{
  _id: ObjectId('68b2b722f1189b6ebcbcd4422'),
  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
}
```

Date :

**Command: db.class10.find()**

```
{
  _id: ObjectId('68b2b48ef1189b6ebcbdd4421'),
  name: 'ramu',
  rollno: 2,
  phone: 123456789
}
{
  _id: ObjectId('68b2b722f1189b6ebcbdd4422'),
  name: 'ramu',
  rollno: 3
}
```

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

```
{
  id: ObjectId('68b2b48ef1189b6ebcbdd4421'),
  name: 'ramu',
  rollno: 2,
  phone: 123456789
}

{
  _id: ObjectId('68b2b722f1189b6ebcbdd4422'),
  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'})`





## 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

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

Date :

**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 })
```

