# *Unit-I Introduction to JavaScript and JavaScript Frameworks*

## Introduction to JavaScript

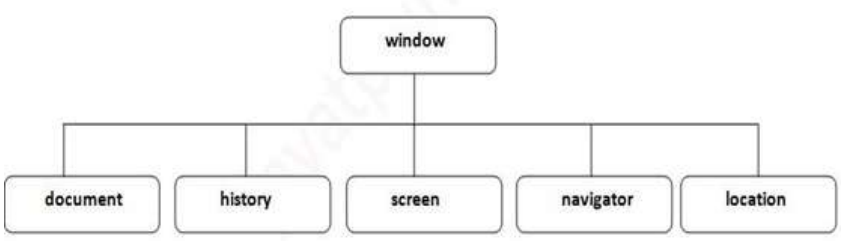
- JavaScript is an object-based scripting language
- Lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- An interpreted, full-fledged programming language enables dynamic interactivity on websites when applied to an HTML document.
- Users can build modern web applications to interact directly without reloading the page every time.
- JavaScript has no connectivity with Java programming language.
- In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

## Features of JavaScript

- All popular web browsers support JavaScript as they provide built-in execution environments.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
- JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
- It is a light-weighted and interpreted language.
- It is a case-sensitive language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.
- It provides good control to the users over the web browsers**.**

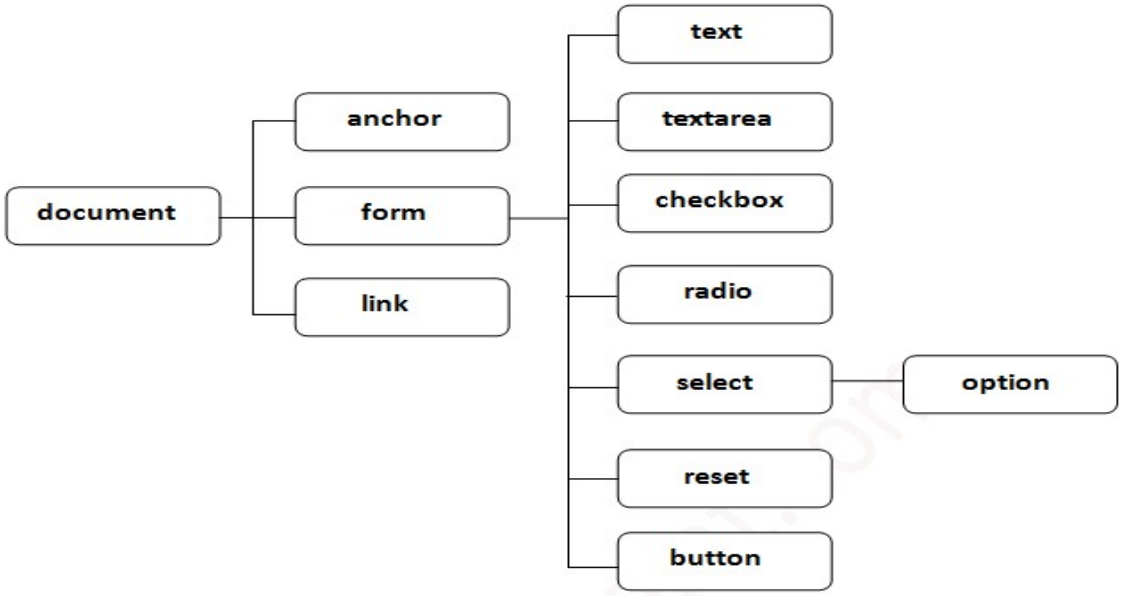
## Browser object model

- The Browser Object Model (BOM) is used to interact with the browser.
- The default object of browser is window means you can call all the functions of window by specifying window or directly.



## Document Object model

- When html document is loaded in the browser, it becomes a document object.
- The root element represents the html document.
- It has properties and methods. By the help of document object, we can add dynamic content to our web page.




## Accessing and manipulating DOM

- Methods
  - getElementByID()
  - getElementsByName()
  - getElementsByTagName()
  - getElementsByClassName()
- Properties
  - innerHTML
  - innerText

## What Is the Arrow Function?

- When you need to create a function in JavaScript, the main method is to use the function keyword followed by the function name as shown below:

```
function greetings(name) {
  console.log('Hello, ${name}!');
}
greetings('John'); // Hello, John!
```

- The arrow function syntax allows you to create a function expression that produces the same result as the code above.
- Here's the greetings() function again, but using the arrow function syntax:

```
const greetings = name => {
  console.log('Hello, ${name}!');
};
greetings('John'); // Hello, John!
```

- When you declare a function with the arrow function syntax, you need to assign the declaration to a variable so that the function has a name.

## Arrow function syntax:

```
const myFunction = (param1, param2, ...) => {
  // function body
}
```

- In the code above, the myFunction is the variable that holds the function. You can call the function as myFunction() later in your code.
- (param1, param2, ...) are the function parameters. You can define as many parameters as required by the function.
- Then you have the arrow => to indicate the beginning of the function. After that, you can write curly brackets {} to indicate the function body, or remove them if you have a single-line function.
- At first, the arrow function may seem weird as you are used to seeing the function keyword. However, as you start using the arrow syntax, you will see that it's very convenient and easier to write.
- Let me show you an easy way to convert a regular function to an arrow function next.

## How to Convert a Regular Function to an Arrow Function Easily

You can follow these **three easy steps** to convert a regular function to an arrow function:

1. Replace the function keyword with the variable keyword const
2. Add the = symbol after the function name and before the parentheses
3. Add the => symbol after the parentheses

Usually, a function is never changed after the declaration, so we use the const keyword instead of let.

The code below should help you visualize the steps:

```
function greetings(name) {
  return 'Hello, ${name}!';
}

// step 1: replace function with const
```

```
const greetings(name) {
 return 'Hello, ${name}!';
}

// step 2: add = after the function name
const greetings = (name) {
 return 'Hello, ${name}!';
}

// step 3: add => after the parentheses
const greetings = (name) => {
 return 'Hello, ${name}!';
}
```

- The three steps above are enough to convert any old JavaScript function syntax to the new arrow function syntax.
- When you have a single line function, there's a fourth optional step to remove the curly brackets and the return keyword as follows:

```
// from this
const greetings = (name) => {
 return 'Hello, ${name}!';
};

// to this
const greetings = (name) => 'Hello, ${name}!';
```

- When you have exactly one parameter, you can also remove the parentheses:

```
// from this
const greetings = (name) => 'Hello, ${name}!';

// to this
const greetings = name => 'Hello, ${name}!';
```

- But the last two steps are optional. Only the first three steps are required to convert any JavaScript function created using the function keyword into the arrow function syntax.

## Why Arrow Functions Are Recommended Over Regular Functions

- The arrow function syntax offers improvements to the way you write a function in JavaScript, such as:
  - You can write short functions in a more straightforward manner
  - For single-line functions, the return statement can be implicit
  - "This" keyword is not bound to the function.
  - Let us see how these improvements work with practical examples next.

## Arrow Functions Are Better for Short Functions

- Suppose you have a single-line function that prints a string to the console. Using the function keyword, here is how you would write the function:

```
function greetings(name) {
 console.log('Hello, ${name}!');
}
```

- If you use the arrow function syntax, you can omit the curly brackets, creating a single-line function as shown below:

```
const greetings = (name) => console.log('Hello, ${name}!');
```

- Even more, you can remove the parentheses that surround the function parameters when you have exactly one parameter:

```
const greetings = name => console.log('Hello, ${name}!');
```

- If your function has no parameter, then you need to pass empty parentheses between the assignment and the arrow syntax as shown below:

```
const greetings = () => console.log('Hello, World!');
```

- When using the arrow function syntax, the curly brackets are required only when your function is more than a single line. For example:

```
const greetings = () => {
 console.log('Hello World!');
 console.log('How are you?');
};
```

## Arrow Functions Have an Implicit Return Statement

- When you have a single-line arrow function, JavaScript will add the return statement implicitly. This means you should not add the return keyword explicitly.
- To show you what I mean, suppose you have a function that sums two numbers as follows:

```
function sum(a, b) {
 return a + b;
}
```

- When you write the function above using the arrow function syntax, you need to remove the curly brackets and the return keyword:

```
const sum = (a, b) => a + b;
```

- If you didn't remove the return keyword, then JavaScript will throw an error, saying an opening curly bracket { is expected.
- When you use arrow functions, only write the return statement explicitly when you have multi-line statements:

```
const sum = (a, b) => {
 const result = a + b;
 return result;
};
```

- When you remove the curly brackets, don't forget to remove the return keyword if you use it.

## JavaScript and JSON

- JSON stands for JavaScript Object Notation. JSON is a text-based data format that is used to store and transfer data. For example,

```
// JSON syntax
{
  "name": "John",
  "age": 22,
  "gender": "male",

}
```

- In JSON, the data are in key/value pairs separated by a comma.
- JSON was derived from JavaScript. So, the JSON syntax resembles JavaScript object literal syntax. However, the JSON format can be accessed and be created by other programming languages too.
- JavaScript Objects and JSON are not the same.

## Use of JSON

- JSON is the most commonly used format for transmitting data (data interchange) from a server to a client and vice-versa. JSON data are very easy to parse and use. It is fast to access and manipulate JSON data as they only contain texts.
- JSON is language independent. You can create and use JSON in other programming languages too.

## JSON Data

- JSON data consists of key/value pairs similar to JavaScript object properties. The key and values are written in double quotes separated by a colon :. For example,

```
// JSON data
"name": "John"
```

- JSON data requires double quotes for the key.

## JSON Object

- The JSON object is written inside curly braces { }. JSON objects can contain multiple key/value pairs. For example,

```
// JSON object
{ "name": "John", "age": 22 }
```

## JSON Array

- JSON array is written inside square brackets [ ]. For example,

```
// JSON array
[ "apple", "mango", "banana"]

// JSON array containing objects
[
  { "name": "John", "age": 22 },
  { "name": "Peter", "age": 20 }.
  { "name": "Mark", "age": 23 }
]
```

- JSON data can contain objects and arrays. However, unlike JavaScript objects, JSON data cannot contain functions as values.

## Accessing JSON Data

- You can access JSON data using the dot notation. For example,

```
// JSON object
const data = {
  "name": "John",
  "age": 22,
  "hobby": {
        "reading" : true,
        "gaming" : false,
        "sport" : "football"
  },
  "class" : ["JavaScript", "HTML", "CSS"]
}

// accessing JSON object
console.log(data.name); // John
console.log(data.hobby); // { gaming: false, reading: true, sport: "football"}

console.log(data.hobby.sport); // football
console.log(data.class[1]); // HTML
```

- We use the . notation to access JSON data. Its syntax is: variableName.key
- You can also use square bracket syntax [] to access JSON data. For example,

```
// JSON object
const data = {
  "name": "John",
  "age": 22
}

// accessing JSON object
console.log(data["name"]); // John
```

## JavaScript Objects VS JSON

- Though the syntax of JSON is similar to the JavaScript object, JSON is different from JavaScript objects.

| JSON | JavaScript Object |
|---|---|
| Text-based format for data exchange. | In-memory data structure in JavaScript. |
| Supports limited types: string, number, boolean, null, array, object. | Supports all JavaScript data types, including undefined. |
| No methods or executable code. | Can include methods and execute logic. |
| JSON can be created and used by other programming languages. | JavaScript objects can only be used in JavaScript. |
| Used for transferring data between systems. | Used for in-program data manipulation. |
| Requires parsing to convert to JavaScript Object | Can be directly used in JavaScript programs. |

## Converting JSON to JavaScript Object

- You can convert JSON data to a JavaScript object using the built-in JSON.parse() function. For example,

```
// json object
const jsonData = '{ "name": "John", "age": 22 }';

// converting to JavaScript object
const obj = JSON.parse(jsonData);

// accessing the data
console.log(obj.name); // John
```

## Converting JavaScript Object to JSON

- You can also convert JavaScript objects to JSON format using the JavaScript built-in JSON.stringify() function. For example,

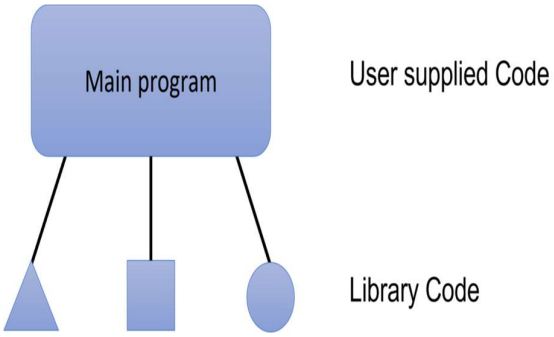
```
// JavaScript object
const jsonData = { "name": "John", "age": 22 };

// converting to JSON
const obj = JSON.stringify(jsonData);

// accessing the data
console.log(obj); // "{"name":"John","age":22}"
```

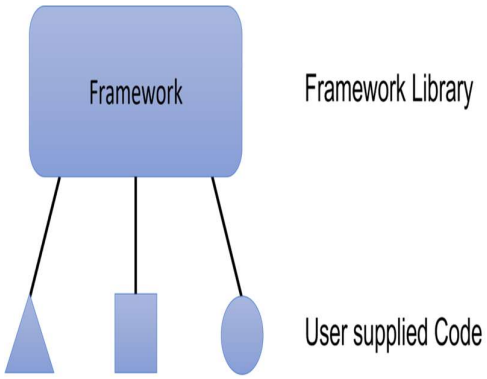
# JavaScript Frameworks & Libraries

## What is Library?




- In contrast to pure code, a library is not an independently executable unit, but an additional module that is requested by a programme.
- libraries are collections of classes and functions, which is why they are sometimes also called component or class libraries. With the help of a programme interface (API / Application Programming Interface), access to the necessary functions from the library is made possible.
- However, access is only limited to the "public" functions; libraries also have "private" functions that work in the background but remain hidden from the outside world.

## What is Framework?




- A framework is a special kind of library that **does not contain "finished" functions** like a regular library.
- Frameworks are rather programme scaffolds that provide the blueprint but not the finished unit.
- The framework, therefore, provides the blueprint and the basic framework and also shows what is still needed from the programmer in terms of customization.
- The **framework provides the flow of a software application** and tells the developer what it needs and calls the code provided by the developer as required. If a library is used, the application calls the code from the library.

*Note: When you use a library, you are in charge of the flow of the application. You are choosing when and where to call the library. When you use a framework, the framework is in charge of the flow.*

**Uses of JavaScript Library**

- DOM Manipulation
- Data Handling
- Animation
- Database
- Maps
- User Interface
- Charts
- UI Components

## JavaScript Library and framework


| Javascript Libraries | JavaScript Frameworks |
|---|---|
| Jquery – Use DOM<br>React.JS – Single page application<br>D3.js – Data Driven Document (Data Visualization)<br>Underscore.js – light weight<br>Anim.js – For animation<br>Chart.js – for designing canvas element<br>And many more<br>Redux - open-source JavaScript library for managing and centralizing application state | Angular – web framework for cross platform application<br>Vue.js – Single page web application<br>Ember.js – used to create resizable client-side web applications.<br>Node.js – Server runtime environment<br>Next.js – react based framework<br>Mocha – Testing framework |

# MEAN.JS introduction

MEAN is an open source JavaScript framework, used for building dynamic websites and web applications. It includes following four building blocks to build an application.

- **MongoDB** – It is a document database, that stores data in flexible, JSON-like documents.
- **Express** – It is web application framework for Nodejs.
- **Node.js** – It is Web Server Platform. It provides rich library of various JavaScript modules which simplifies the development of web applications.
- **AngularJS** – It is a web frontend JavaScript framework. It allows creating dynamic, single page applications in a clean Model View Controller (MVC) way.

The term MEAN.js is a full stack JavaScript open-source solution, used for building dynamic websites and web applications. MEAN is an acronym that stands for MongoDB, Express, Node.js and AngularJS, which are the key components of the MEAN stack.

It was basically developed to solve the common issues with connecting those frameworks (Mongo, Express Nodejs, AngularJS), build a robust framework to support daily development needs, and help developers use better practices while working with popular JavaScript components.

Stack means using the database and web server in the back end, in the middle you will have logic and control for the application and interaction of user at the front end.

- MongoDB – Database System
- Express – Back-end Web Framework
- Node.js – Web Server Platform
- AngularJS – Front-end Framework

## MongoDB

- It is a schema less NoSQL database system. MongoDB saves data in binary JSON format which makes it easier to pass data between client and server.
- Uses a JSON document that is relevant to the data model.
- A schema-free, self-contained NoSQL database
- Highly scalable design designed to process massive amounts of data that is both cost-effective and beneficial in transporting data between the server and a client
- Assists with file storage, index preparation, and bandwidth allocation.

## ExpressJS

- The server-side JavaScript framework is a minimalist framework. It is lightweight framework used to build web applications in Node. It provides a number of robust features for building single and multi-page web application. Express is inspired by the popular Ruby framework, Sinatra.

- Reduces the amount of work involved in developing secure online and mobile apps.
- Developers can add additional innovations and developments.
- Back-end development is mostly done with this framework.
- It is easier to develop server-side apps with Nodejs.
- Database connectivity, template processors, and multiple basic routing options.

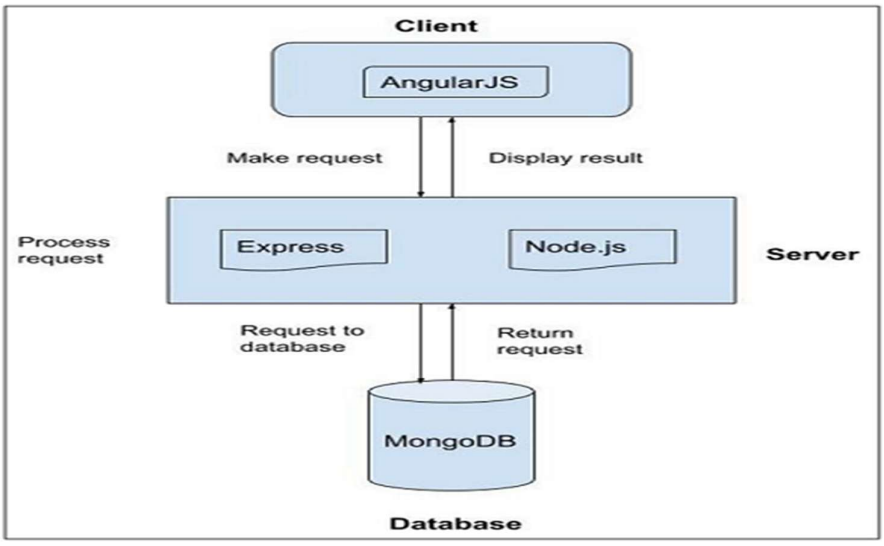
## AngularJS

- A JavaScript framework for online front-end development.
- Google maintains and builds a computer MVC JavaScript data conditional UI framework that is flexible in structure, development, and testing.
- Reduce the time it takes for dynamic and SPAs to emerge.
- For a full-stack web front-end framework, it's easy to learn and scale.

## NodeJS

- A JavaScript-based execution framework that is open-source and cross-platform.
- Built using the JS V8engine from Google Chrome
- Aids in the development of feature-rich real-time web-based applications.
- Before the operation, convert JavaScript software to native machine code.
- It enables the creation of server-based applications that are both accessible and safe.
- It has a diverse ecosystem of open-source modules and features.

## Architecture of MEAN.JS.




**Three-Layer Architecture of MEAN.js**

MEAN.js follows a three-layer architecture consisting of the Client Layer, Application Layer, and Data Layer. Each layer has a specific role, making the application modular, scalable, and easy to maintain.

**1. Client Layer (Presentation Layer – AngularJS)**

The **Client Layer** is responsible for the **presentation and user interaction** part of the application. It runs in the user's web browser.

Key Responsibilities:

- Displays web pages and user interfaces.
- Handles user input such as forms, clicks, and events.
- Performs client-side validation.
- Communicates with the server using **RESTful HTTP requests**.
- Dynamically updates the view without reloading the page.

Technologies Used:

- **AngularJS (JavaScript)**
- HTML and CSS

Architectural Pattern:

- Follows **MVC (Model-View-Controller)** architecture.
- Uses controllers, directives, and services.
- Two-way data binding between Model and View.

Data Flow:

- Sends HTTP requests (GET, POST, PUT, DELETE) to the Express server.
- Receives JSON responses and binds data to the UI.

**2. Application Layer (Business Logic Layer – Node.js & Express.js)**

The **Application Layer** acts as the bridge between the client and the database. It contains the **business logic** of the application.

Key Responsibilities:

- Handles client requests and responses.
- Implements business rules and workflows.
- Manages authentication and authorization.
- Validates input data and handles errors.
- Defines RESTful APIs.
- Communicates with MongoDB for data operations.

Technologies Used:

- **Node.js** – Server-side JavaScript runtime
- **Express.js** – Web application framework

Components:

- Routes
- Controllers
- Middleware
- Services

Data Flow:

- Receives requests from AngularJS.
- Processes data using Node.js logic.
- Interacts with MongoDB via models.
- Sends JSON responses back to the client.

**3. Data Layer (Database Layer – MongoDB)**

The **Data Layer** is responsible for **data storage and management** in the MEAN stack.

Key Responsibilities:

- Stores application data permanently.
- Performs CRUD operations.
- Maintains data consistency.
- Supports data indexing and aggregation.

Technologies Used:

- **MongoDB**
- **Mongoose** (Object Data Modeling library)

Data Structure:

- Uses **collections and documents** instead of tables.
- Data is stored in **BSON (Binary JSON)** format.

Features:

- Schema-less or flexible schema design.
- Supports horizontal scaling.
- Provides high availability through replication.