React Basic Concepts

What is React js?

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies.

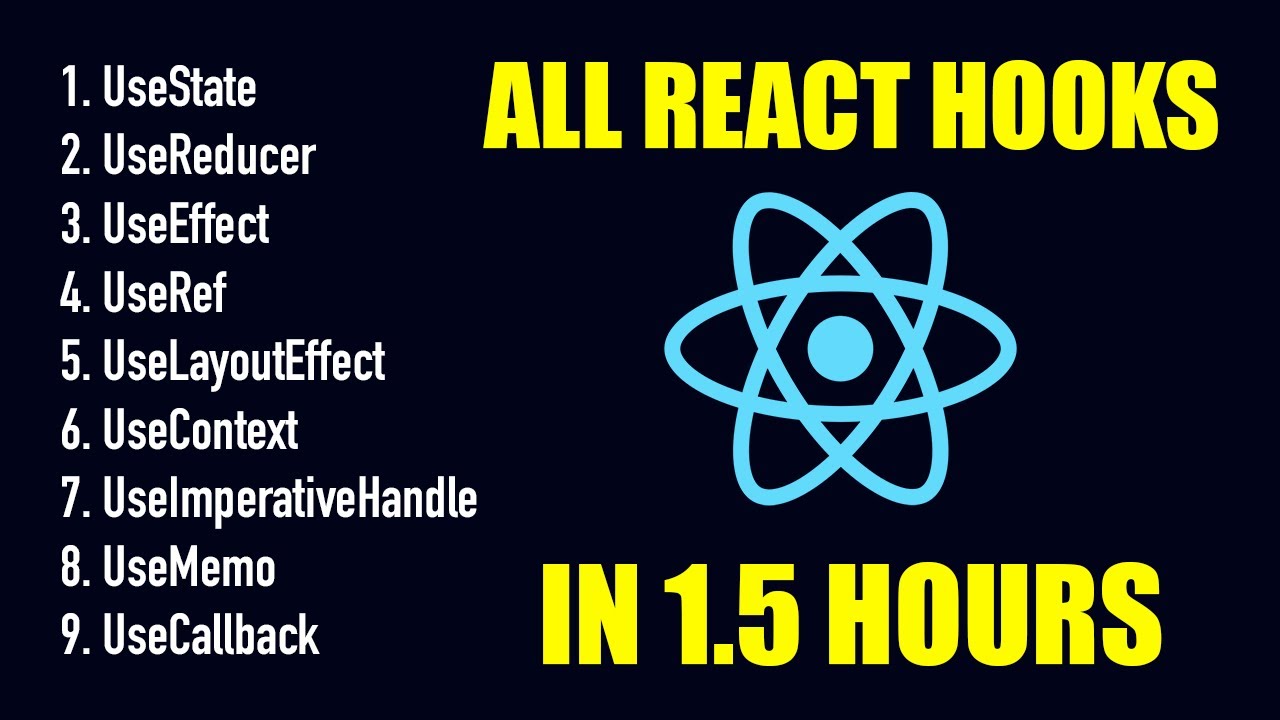
**What is React State?**

The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders

**What is React Hooks?**

A React hook is a function in React that allows developers to use state and other React features in functional components.

React Hooks are simple JavaScript functions that we can use to isolate the reusable part from a functional component. Hooks can be stateful and can manage side-effects. React provides a bunch of standard in-built hooks: useState : To manage states. Returns a stateful value and an updater function to update it



**What is useState Hook?**

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

**What is useEffect?**

The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

**What is UseRef?**

It can be used to store a mutable value that does not cause a re-render when updated.

**What is useContext?**

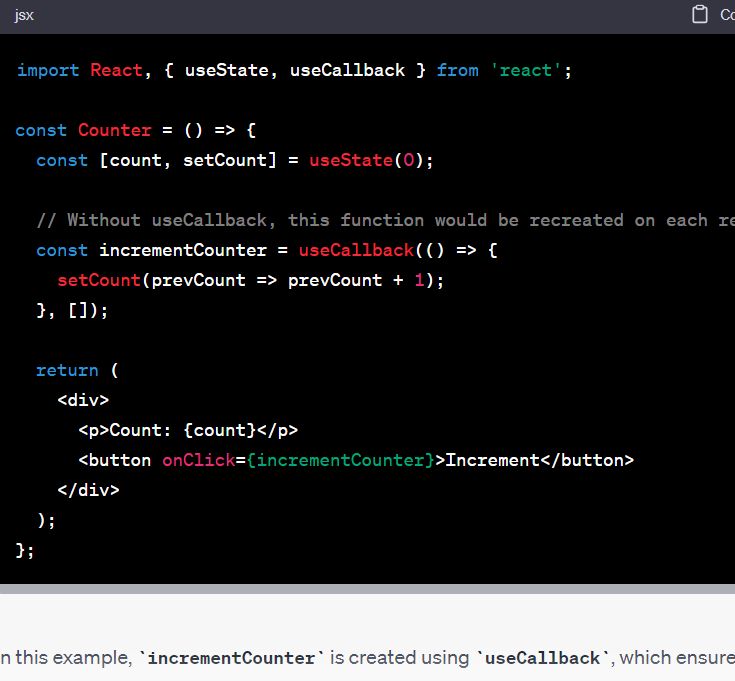
React Context is a way to manage state globally.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

With useContext we cannot use prop drilling i.e passing data as props from component to component.

**What is useCallback?**

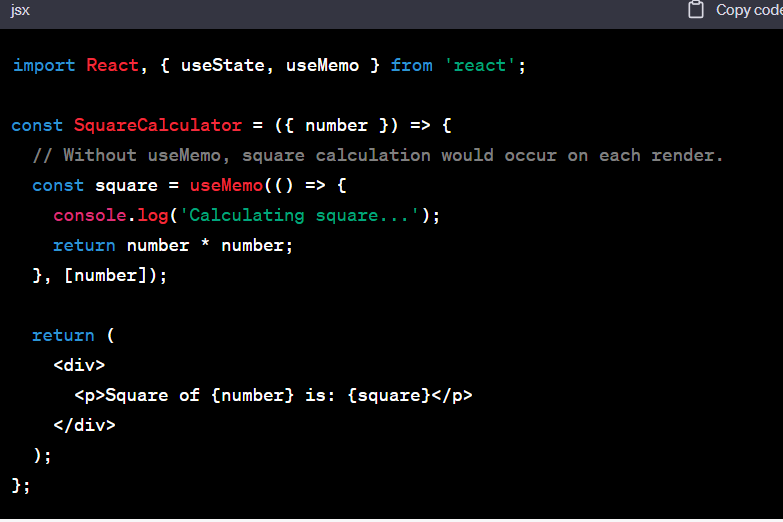
The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function. You can learn more about useMemo in the useMemo [chapter](https://www.w3schools.com/react/react_usememo.asp).



**What is useMemo?**

The React useMemo Hook returns a memoized value.

Think of memoization as caching a value so that it does not need to be recalculated.



**What is Component?**

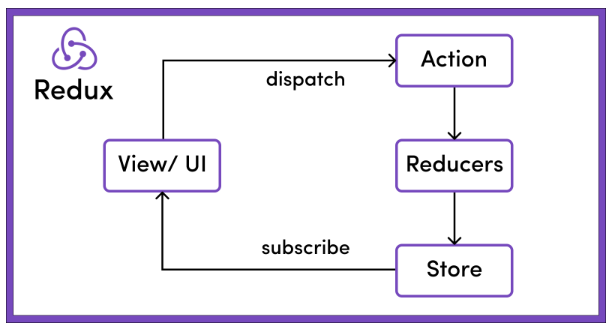
Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Components come in two types, Class components and Function components.

**What is Redux?**

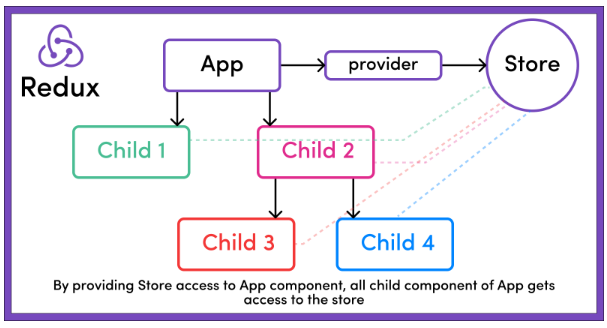
it's mostly used as a state management tool with React.

Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces



The Redux store is the main, central bucket which stores all the states of an application.

If the store is provided to the App.js (by wrapping the App component within the <Provider> </Provider> tag) as shown in the code snippet below, then all its children (children components of App.js) can also access the state of the application from the store. This makes it act as a global state.



The only way to change the state is to emit an action, which is an object describing what happened.

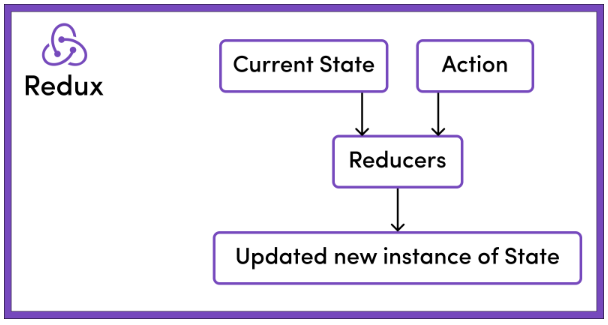
Let's take the example of the above store example where we have 2 books in the store: "Harry Potter and the Chamber of Secrets" and "Harry Potter and the Prisoner of Azkaban". There's just one copy of each.

Now if the user wants to add another item to the cart, then they will have to click on the "Add to Cart" button next to the item.

On the click of the "Add to Cart" button, an action will be dispatched. This action is nothing but a JS object describing what changes need to be done in the store.

**What Are Reducers in Redux?**

To specify how the state tree is transformed by actions, we write pure reducers.



Reducers, as the name suggests, take in two things: previous state and an action. Then they reduce it (read it return) to one entity: the new updated instance of state.

So reducers are basically pure JS functions which take in the previous state and an action and return the newly updated state.

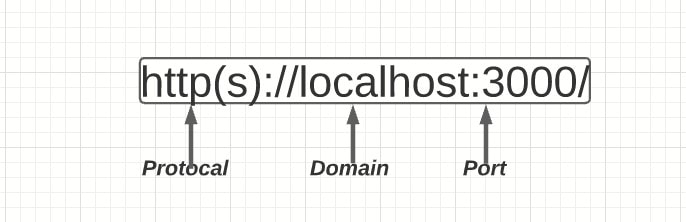
**Difference between fetch and axios?**

Fetch is supported by every browser while axios is a npm package in which every browser not support it.

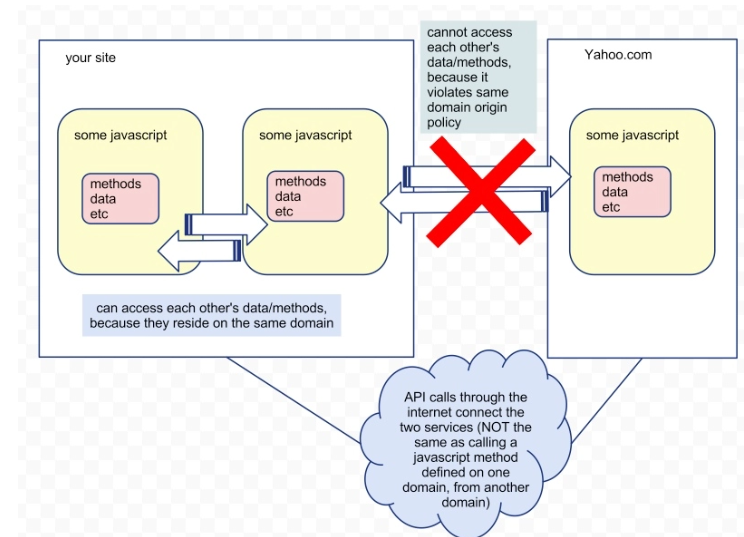
In axios sending data is in already json format while in fetch we himself convert data to json and than send it.

**What is CORS?**

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.



The same-origin policy is a critical security mechanism that restricts how a document or script loaded from one origin can interact with a resource from another origin. It helps isolate potentially malicious documents, reducing possible attack vectors.



**Lifecycle of React Components**

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: **Mounting**, **Updating**, and **Unmounting**.

**Mounting**

React has four built-in methods that gets called, in this order, when mounting a component:

* **constructor()**
* **getDerivedStateFromProps()**
* **render()**
* **componentDidMount()**

The **render**() method is required and will always be called, the others are optional and will be called if you define them.

**constructor**

The **constructor**() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

**getDerivedStateFromProps**

The **getDerivedStateFromProps**() method is called right before rendering the element(s) in the DOM.

**render**

The **render**() method is required, and is the method that actually outputs the HTML to the DOM.

**componentDidMount**

The **componentDidMount**() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

**Updating**

The next phase in the lifecycle is when a component is updated.

A component is updated whenever there is a change in the component's **state** or **props**.

Also at updates the **getDerivedStateFromProps** method is called. This is the first method that is called when a component gets updated.

The **render**() method is of course called when a component gets updated, it has to re-render the HTML to the DOM, with the new changes.

**componentDidUpdate**

The **componentDidUpdate** method is called after the component is updated in the DOM.

**What is ES6?**

ES6 stands for ECMAScript 6.

ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

**React ES6 Spread Operator**

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

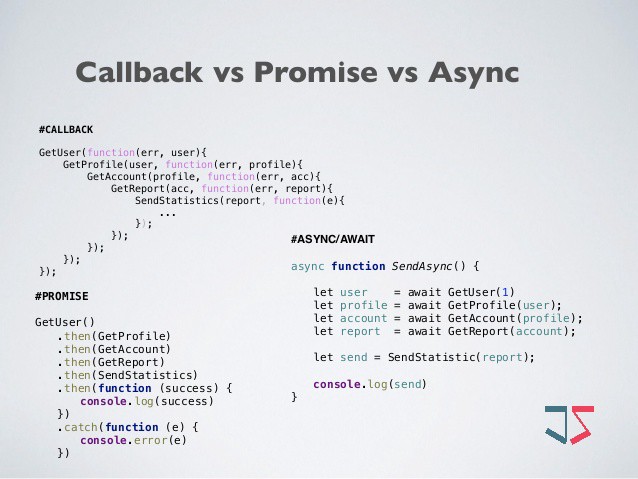
const numbersOne = [1, 2, 3];

const numbersTwo = [4, 5, 6];

const numbersCombined = [...numbersOne, ...numbersTwo];

**OUTPUT**: [1,2,3,4,5,6]

**What is promise in node js?**

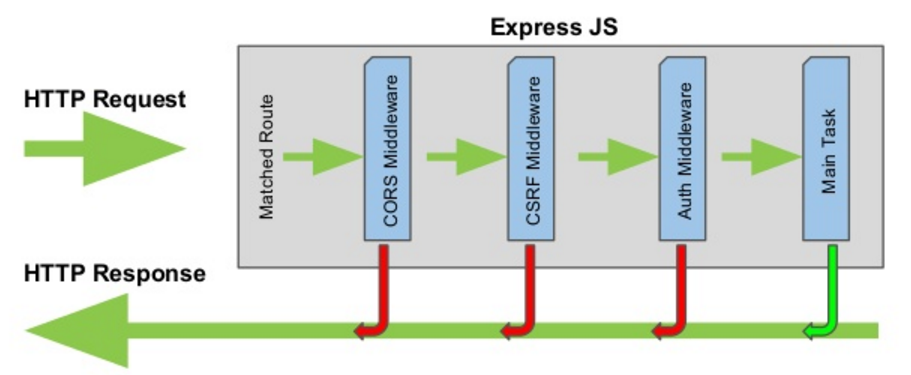
Promise is a placeholder for a value that will be available in the future, allowing us to handle the result of an asynchronous task once it has completed or encountered an error.

**What is CallBack Function ?**

Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function.

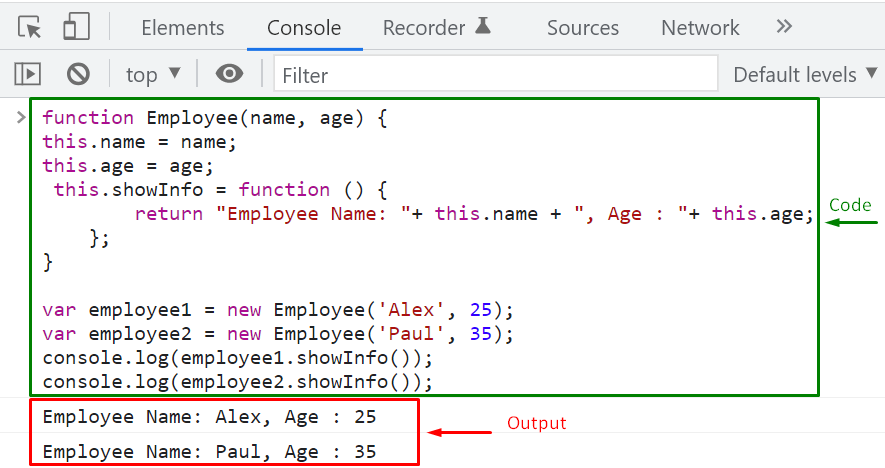
**What is middleware in node js?**

The middleware in node. js is a function that will have all the access for requesting an object, responding to an object, and moving to the next middleware function in the application request-response cycle.



**What is Constructor ?**

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the new keyword. The purpose of a constructor is to create a new object and set values for any existing object properties.



**What is destructor?**

The purpose of a destructor is to allow the item that registered the listeners to unregister them. Once an object has no other references to it, it will be garbage collected. For instance, in AngularJS, when a controller is destroyed, it can listen for a destroy event and respond to it.

**What are props?**

We use props in React to pass data from one component to another (from a parent component to a child component(s)). Props is just a shorter way of saying properties. They are useful when you want the flow of data in your app to be dynamic.

**What is node js?**

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows you to run JavaScript on the server side, which can be useful for building servers and backend applications. Node.js is designed to be lightweight and efficient, and it's commonly used for building real-time applications such as chat apps, collaborative code editors, and online gaming. It also has a large ecosystem of open source libraries that can be easily integrated into a Node.js project.

**What is Async middleware?**

async middleware is software that enables asynchronous processing, which allows different components of a system to perform tasks concurrently rather than sequentially. This can improve the overall performance and responsiveness of the system, especially when it needs to handle large amounts of data or perform computations that take a long time to complete. Async middleware typically provides mechanisms for scheduling and executing tasks asynchronously, as well as for communication and coordination between tasks.

**What is Chrome V8 enginer?**

Chrome's V8 JavaScript engine is an open-source, high-performance JavaScript engine that powers the Google Chrome web browser and other applications. It was created by Google's V8 team, and it is written in C++.

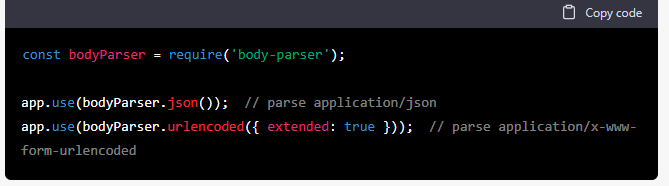
The V8 engine compiles JavaScript code into machine code instead of interpreting it on the fly, which allows for faster performance. It uses a just-in-time (JIT) compilation technique to optimize the code as it runs, identifying hot spots and optimizing them for faster execution.

The V8 engine is designed to be embedded in other applications as well, and it has been used in projects such as Node.js, Deno, and Electron. It provides a rich set of APIs that allow developers to interact with the engine and customize its behavior.

One of the key features of the V8 engine is its support for modern JavaScript language features, such as async/await and destructuring. It also supports new features as they are added to the language, which helps keep JavaScript up-to-date with the latest programming trends.

**What is body parser in node js?**

The body-parser middleware is a popular package for Node.js that parses the body of an incoming request and makes it easier to work with. It can parse the body of a request in various formats, such as JSON, URL-encoded data, and raw data, and makes the parsed data available in the request object for your application to use.



**what is config.json in node js?**

config.json is a file that contains configuration settings for an application. It is similar to a config.js file, but it is typically formatted as a JSON object rather than a JavaScript module. Like config.js, config.json is

Often used to store settings that are specific to the environment in which the application is running, such as the database connection parameters or the application's base URL.

**OOP CONCEPTS**

**What is Inheritance?**

In object-oriented programming (OOP), inheritance is a way to create a new class that is a modified version of an existing class. The new class is called the subclass, and the existing class is the superclass. The subclass inherits the attributes and behaviors of the superclass, and can also have additional attributes and behaviors of its own.



**What is Polymorphism?**

Polymorphism is a fundamental concept in object-oriented programming (OOP) that refers to the ability of a single interface to be used to access a variety of different implementations. In other words, it allows you to write code that can work with multiple types of objects in a uniform way, even if those objects have different underlying implementations.



**What is encapsulation?**

In object-oriented programming (OOP), encapsulation is the concept of bundling data and methods that operate on that data within a single unit, or object. This is achieved by creating a class that contains the data and methods, and then creating instances of that class, or objects, which contain their own copies of the data and can access the methods.

For example, consider a class **Person** that represents a person with a **name** and a **age**. The class might have a private member variable age to store the person's age, and public methods **getAge**() and **setAge**(int age) to allow external code to access and modify the person's age. By making the age variable private and providing access to it only through the **getAge**() and **setAge**(int age) methods, the class can enforce rules on how the age can be accessed and modified, such as ensuring that the age is always a positive integer.

**What is Abstraction in OOP?**

In object-oriented programming (OOP), abstraction is the process of exposing only the essential characteristics of an object, while hiding its implementation details.

Abstract classes are classes that cannot be instantiated and are used to provide a common interface or set of behaviors for a group of related classes.

An abstract class can contain both abstract methods, which are methods that have a declaration but no implementation, and concrete methods, which have both a declaration and an implementation. Subclasses of the abstract class must implement the abstract methods of the parent class, providing their own implementations of the methods.

**What is interface in OOP?**

In object-oriented programming (OOP), an interface is a type of class that defines a set of abstract methods that must be implemented by any class that implements the interface.

For example, consider an interface **Shape** that defines abstract methods **area**() and **perimeter**() for calculating the **area** and **perimeter** of a **shape**. A class **Circle** that implements the **Shape** interface would need to provide its own implementation of the **area**() and **perimeter**() methods, but it could then be used in the same way as any other class that implements the **Shape** interface, such as a **Rectangle** class.

**What is Typescript?**

TypeScript is a programming language developed and maintained by Microsoft. It is a typed superset of JavaScript, meaning that it adds optional static typing and class-based object-oriented programming to the dynamic and functional capabilities of JavaScript.

**Difference between Typescript and Javascript?**

One of the main differences between TypeScript and JavaScript is that TypeScript is a compiled language, while JavaScript is an interpreted language. This means that TypeScript code is transformed into JavaScript code by a compiler before it is run in a web browser or other runtime environment, while JavaScript code is executed directly by the runtime environment.

TypeScript also has additional language features that are not available in JavaScript, such as classes, interfaces, and modules. These features can make it easier to structure and organize code, especially in large projects.

**What is data modifier?**

A data modifier, also known as an access modifier or access specifier, is a keyword in a programming language that is used to control the visibility and accessibility of class members (variables and methods) from outside the class.

There are generally four types of data modifiers: public, private, protected, and package (or default). The specific meaning and behavior of these modifiers can vary depending on the programming language, but they generally have the following meanings:

* **public**: Members marked as public can be accessed from anywhere within the program.
* **private**: Members marked as private can only be accessed from within the class in which they are declared.
* **protected**: Members marked as protected can be accessed from within the class in which they are declared and from any subclass of that class.
* **package** (or default): Members marked with no access modifier are said to have package or default access and can be accessed from within the package in which they are declared, but not from outside the package.

**What is lazy and eager loading?**

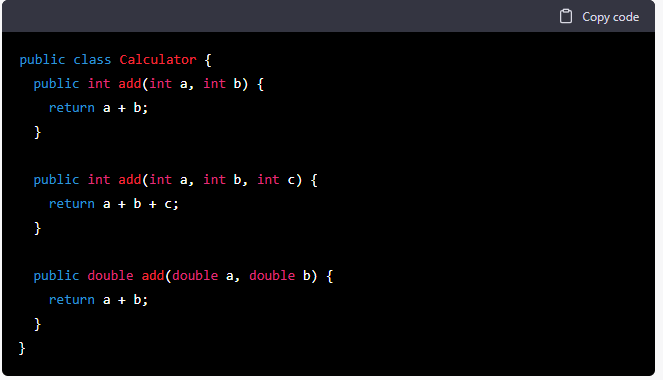
**Lazy** loading is a technique in which resources are only loaded when they are needed, rather than being loaded all at once when the page or application is first loaded. This can help to improve the initial load time of a page or application by only loading the resources that are immediately required, and deferring the loading of other resources until they are needed.

For example, consider a website that displays a list of images. Using lazy loading, the images would only be loaded when they are scrolled into view, rather than being loaded all at once when the page is first loaded. This can help to reduce the amount of data that needs to be transferred and processed when the page is first loaded, improving the initial load time of the page.

**Eager** loading is the opposite of lazy loading, and involves loading all of the resources required by a page or application as soon as possible, regardless of whether they are immediately needed. This can be useful in some cases, such as when resources are needed very soon after the page is loaded, but it can also result in slower initial load times if there are a large number of resources to be loaded.

**What is method overloading?**

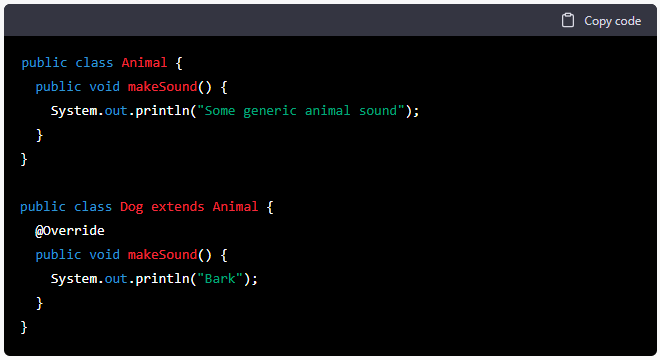
Method overloading is a feature in object-oriented programming languages that allows a class to have multiple methods with the same name, but with different numbers or types of parameters.



**What is method overriding?**

Method overriding is a feature in object-oriented programming languages that allows a subclass to provide a different implementation of a method that is inherited from a parent class.

The idea behind method overriding is to allow a subclass to customize or extend the behavior of a method inherited from a parent class, while still maintaining the same interface as the parent class. This is often done to specialize the behavior of the method for the specific needs of the subclass.



**How to find Second Larget Number in Array?**

const arr = [3,2,6,7,8];

function second\_larget\_find(arr)

{

  largest = arr[0];

  second\_larget = arr[1] ;

  for(var i=2; i<arr.length; i++)

  {

    if(arr[i] > largest)

    {

      second\_larget = largest;

      largest = arr[i];

    }

    else if(arr[i] > second\_larget)

    {

      second\_larget = arr[i];

    }

  }

  console.log(second\_larget)

}

second\_larget\_find(arr);

**Print Star in Pyramid Shape**

function printPyramid(height) {

// Iterate through each row

for (let i = 1; i <= height; i++) {

let row = '';

// Print appropriate number of spaces before stars for each row

for (let j = 1; j <= height - i; j++) {

row += ' ';

}

// Print appropriate number of stars after spaces for each row

for (let k = 1; k <= 2 \* i - 1; k++) {

row += '\*';

}

console.log(row);

}

}

// Call the function with a height of 5

printPyramid(5);

**Print star in Triangle Shape?**

function printTriangle(height) {

// Iterate through each row

for (let i = 1; i <= height; i++) {

let row = '';

// Print appropriate number of stars for each row

for (let j = 1; j <= i; j++) {

row += '\*';

}

console.log(row);

}

}

// Call the function with a height of 5

printTriangle(5);

**what is event loop in node js?**

In Node.js, the event loop is a mechanism that allows for asynchronous programming by constantly monitoring the execution stack for any tasks that need to be executed. The event loop continuously processes events and callbacks from the event queue, and schedules them for execution in the order they were received.

When an asynchronous operation is initiated in Node.js, it is passed to the event loop for processing, and the event loop schedules it for execution in the future. The event loop allows Node.js to continue processing other requests while it waits for the asynchronous operation to complete.

In other words, the event loop is a crucial part of Node.js that allows for non-blocking I/O operations and efficient use of system resources. It is responsible for managing all I/O operations, callbacks, and timers in a single thread, ensuring that Node.js can handle a large number of concurrent requests without consuming excessive resources or experiencing performance issues.

## **Phases of a React component's lifecycle**

A React component undergoes three phases in its lifecycle: **mounting**, **updating**, and **unmounting**.

1. The mounting phase is when a new component is created and inserted into the DOM or, in other words, when the life of a component begins. This can only happen once, and is often called “initial render.”
2. The updating phase is when the component updates or re-renders. This reaction is triggered when the props are updated or when the state is updated. This phase can occur multiple times, which is kind of the point of React.
3. The last phase within a component's lifecycle is the unmounting phase, when the component is removed from the DOM.

**what is chrome v8 engine?**

Chrome V8 is an open-source JavaScript engine developed by the Google Chrome team. It is written in C++ and is used in the Google Chrome browser, as well as other web browsers such as Opera and Node.js.

The V8 engine is designed to run JavaScript code quickly and efficiently, and uses a Just-In-Time (JIT) compiler to optimize code at runtime. It compiles JavaScript code into native machine code, which allows it to execute more quickly than interpreted code.

V8 provides a number of features that help make JavaScript execution faster and more efficient, including:

* Garbage collection: V8 uses a garbage collector to automatically free up memory that is no longer being used by a program.
* Hidden classes: V8 uses a technique called hidden classes to optimize the performance of JavaScript objects and properties.
* Inline caching: V8 uses inline caching to speed up the execution of frequently called functions.

**Difference between npm and npx?**

NPM (Node Package Manager) is the default package manager for Node.js and is used to install and manage packages and dependencies for a project. It allows developers to specify and manage project dependencies in a package.json file and install them via the command line.

NPX, on the other hand, is a tool that comes bundled with NPM, but it is used for executing packages without the need to install them. It allows developers to run commands and tools from a package without installing it globally or locally. NPX is particularly useful for running packages that are used only once or infrequently.

**what is real DOM?**

The real DOM (Document Object Model) is a programming interface used by web browsers to render and display HTML, CSS, and JavaScript on web pages. It is a tree-like structure of nodes that represents the elements, attributes, and content of a web page.

When a web page is loaded, the browser creates the real DOM by parsing the HTML and CSS code and constructing a tree of nodes that represents the page's structure. JavaScript can then be used to access and manipulate the nodes in the DOM, allowing developers to dynamically update the content and behavior of the page without reloading it.

**What is Virtual DOM?**

The virtual DOM (Document Object Model) is a concept used by web frameworks and libraries, such as React.js, to improve the efficiency and performance of updating the user interface of a web application.

Instead of directly updating the real DOM, which can be slow and resource-intensive, a virtual representation of the DOM is created in memory, which is then used to track changes to the user interface. When changes are made, the virtual DOM is updated, and then compared to the previous version of the virtual DOM to determine the minimal set of changes that need to be made to the real DOM to update the user interface.

**Explain the concept of middleware in Express.js.**

Middleware functions in Express.js are functions that have access to the **request** and **response** objects and the next middleware function in the application's request-response cycle. Middleware functions can modify request/response objects, execute code, and decide whether to continue to the next middleware or terminate the request-response cycle.

**Explain the concept of a "Callback Hell" and how to avoid it?**

"Callback Hell" occurs when multiple nested callbacks make the code difficult to read and maintain. It happens when dealing with multiple asynchronous operations. To avoid Callback Hell, use techniques like modularization, Promises, or async/await to flatten the code structure and improve readability.

**Explain the difference between process.nextTick() and setImmediate() in Node.js.**

* **process.nextTick()**: It queues a callback to be executed right after the current operation completes, before the event loop continues. This allows a high-priority execution of the callback.
* **setImmediate()**: It queues a callback to be executed in the next iteration of the event loop, after I/O events and any pending callbacks. It provides a way to schedule a lower-priority callback.

**Difference b/w ForEach and Map:**

Foreach cannot create new array and iterates each element of array.

const fruits = ['apple', 'banana', 'orange', 'grape'];

fruits.forEach((fruit) => {

console.log(fruit);

});

// Output:

// apple

// banana

// orange

// grape

In this example, **map** iterates over the **numbers** array, squares each element, and returns a new array **squaredNumbers** with the squared values.

Remember that **forEach** is used for performing actions, such as logging or updating external states, and it doesn't return a new array. On the other hand, **map** is used for transforming the elements and creating a new array with the transformed values.

**Difference between slice and splice in javascirpt ?**

**slice():** This method returns a shallow copy of a portion of an array into a new array, leaving the original array unchanged.

array.slice(startIndex, endIndex);

**startIndex (optional):** The index at which to begin the extraction. If omitted, the default is 0. If negative, it represents an offset from the end of the array.

**endIndex (optional):** The index before which to end the extraction. slice() extracts up to, but not including, the endIndex. If omitted, it goes up to the end of the array. If negative, it represents an offset from the end of the array.

**Example:**

*const fruits = ['apple', 'orange', 'banana', 'grape', 'pear'];*

*const slicedFruits = fruits.slice(1, 4); // Extracts elements at index 1, 2, and 3 (not including 4)*

*console.log(slicedFruits); // Output: ['orange', 'banana', 'grape']*

*console.log(fruits); // The original 'fruits' array remains unchanged.*

**Difference between shift and unshift?**

The **shift()** method is used to remove the first element from an array and return that element. It also updates the original array by removing the element from it.

**array.shift();**

**unshift()** method: The **unshift()** method is used to add one or more elements to the beginning of an array. It updates the original array and returns the new length of the array after adding the elements.

*const fruits = ['banana', 'orange'];*

*const newLength = fruits.unshift('apple', 'grape');*

*console.log(newLength); // Output: 4*

*console.log(fruits); // Output: ["apple", "grape", "banana", "orange"]*

**splice()**: This method is used to add or remove elements from an array, modifying the original array.

**array.splice(startIndex, deleteCount, item1, item2, ...);**

**startIndex**: The index at which to start changing the array. If negative, it represents an offset from the end of the array.

**deleteCount** (optional): The number of elements to remove from the array, starting at the **startIndex**. If set to 0, no elements are removed. If omitted, all elements from **startIndex** to the end of the array will be removed.\

**item1, item2, ...** (optional): Elements to be added to the array starting from the **startIndex.**

**Example:**

*const colors = ['red', 'green', 'blue', 'yellow'];*

*// Remove two elements starting from index 1*

*const removedElements = colors.splice(1, 2);*

*console.log(removedElements); // Output: ['green', 'blue']*

*console.log(colors); // The 'colors' array is now ['red', 'yellow']*

*// Add elements at index 1 without removing any elements*

*colors.splice(1, 0, 'orange', 'purple');*

*console.log(colors); // Output: ['red', 'orange', 'purple', 'yellow']*

**what are Type coercion in javascript.**

Type coercion in JavaScript refers to the automatic conversion of one data type to another during certain operations. It occurs when you use a value of one type in a context that expects a different type. JavaScript tries to make sense of the operation and convert the value to the appropriate type.

Type coercion can be implicit, where JavaScript performs the conversion automatically, or explicit, where you explicitly convert a value from one type to another using functions or operators like **Number()**, **String()**, **Boolean()**, etc.

**Example:**

*// Number + String*

*const num1 = 5;*

*const str1 = "10";*

*const result1 = num1 + str1; // Implicit coercion: '5' + '10' => '510'*

*console.log(result1);*

*// Boolean + Number*

*const bool1 = true;*

*const num2 = 7;*

*const result2 = bool1 + num2; // Implicit coercion: true is converted to 1 => 1 + 7 => 8*

*console.log(result2);*

**Closures in Javascript:**

Closures in JavaScript are a powerful and fundamental concept. A closure is a function that "remembers" its lexical scope even when it is executed outside that scope. In simpler terms, a closure allows a function to access variables from its parent scope even after the parent function has finished executing.

Closures are created when an inner function is returned from an outer function. The inner function retains access to the variables, parameters, and other declarations in its outer function's scope, making them available even when the outer function has completed execution.

Here's an example of a closure in JavaScript:

***function outerFunction() {***

***const outerVariable = 'I am from the outer function';***

***function innerFunction() {***

***console.log(outerVariable);***

***}***

***return innerFunction;***

***}***

***const closureFunction = outerFunction();***

***closureFunction(); // Output: "I am from the outer function"***

**High Order function in javascript:**

A higher-order function (HOF) is a function that takes one or more functions as arguments or returns a new function as its result. In JavaScript, functions are first-class citizens, meaning they can be treated as values and passed around like any other data type. This characteristic allows the creation and usage of higher-order functions

*function doSomething(callback) {*

*console.log('Doing something...');*

*callback(); // Call the callback function*

*}*

*function onComplete() {*

*console.log('Task completed!');*

*}*

*doSomething(onComplete); // Output: "Doing something..." "Task completed!"*

**What are high order components in react js:**

High order functions are functions that take other functions as arguments, or return functions as a result, or do both.

*function map(array, transform) {*

*const result = [];*

*for(let i = 0; i < array.length; i++) {*

*result.push(transform(array[i]));*

*}*

*return result;*

*}*

*const square = function(x) {*

*return x \* x;*

*};*

*const numbers = [1, 2, 3, 4, 5];*

*console.log(map(numbers, square)); // Output: [1, 4, 9, 16, 25]*

**What are reduce in array method? give me example:**

The **reduce()** method in JavaScript is an array method that is used to reduce the elements of an array into a single value. It iterates over the array's elements and accumulates the values based on a callback function provided as an argument. The callback function takes four parameters: accumulator, currentValue, currentIndex, and the array itself.

The **reduce()** method is useful when you need to perform calculations on array elements, concatenate strings, or transform an array into a single value.

*Example:*

*const numbers = [1, 2, 3, 4, 5];*

*const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue, 0);*

*console.log(sum); // Output: 15*

**Difference between var,let and const in javascript:**

**var**:

* Variables declared with **var** have function scope or global scope, depending on whether they are declared inside a function or at the top level of a script.
* They are hoisted to the top of their scope, meaning you can access them before the actual declaration.
* **var** allows redeclaration and reassignment.

**let**:

* Variables declared with **let** have block scope, meaning they are limited to the block in which they are declared (e.g., inside a function or inside curly braces **{}**).
* They are also hoisted, but they are in a "temporal dead zone" before their declaration, meaning you cannot access them before the actual declaration.
* **let** does not allow redeclaration, but it allows reassignment.

**const**:

* Variables declared with **const** have block scope, similar to **let**.
* They must be initialized with a value at the time of declaration and cannot be reassigned after that.
* Like **let**, **const** is also hoisted but in the temporal dead zone before the actual declaration.

### What are the different data types present in javascript?

1. **Primitive types**

* **String**
* **Number**
* **Boolean**
* **BigInt**
* **Float**

1. **Non-primitive types**

Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.

Object - Used to store collection of data.  
**var** obj1 = {

x: 43,

y: "Hello world!",

z: **function**(){

**return** this.x;

}

}

### Explain Hoisting in javascript.

Hoisting is the default behaviour of javascript where all the variable and function declarations are moved on top.

#### ****Note - Variable initializations are not hoisted, only variable declarations are hoisted:****

### Why do we use the word “debugger” in javascript?

The debugger for the browser must be activated in order to debug the code. Built-in debuggers may be switched on and off, requiring the user to report faults. The remaining section of the code should stop execution before moving on to the next line while debugging.

### What is NaN property in JavaScript?

NaN property represents the **“Not-a-Number”**value. It indicates a value that is not a legal number.

**typeof**of NaN will return a **Number**.

To check if a value is NaN, we use the **isNaN()**function,

#### Note- isNaN() function converts the given value to a Number type, and then equates to NaN.

isNaN("Hello") // Returns true

isNaN(345) // Returns false

isNaN('1') // Returns false, since '1' is converted to Number type which results in 0 ( a number)

**Shallow Copy**: A shallow copy means creating a new object/array that stores references to the original object's properties or elements, not the actual objects themselves. If the property is a primitive type (like a number or string), it copies the value. But if it's a reference type (like an object or array), it copies the reference, not the actual object or array. This means changes to nested objects or arrays will be reflected in both the original and the copied object.

**Deep Copy**: A deep copy means creating a new object/array and recursively copying all objects found in the original into the new one. This way, the new object/array is a completely independent clone of the original, and changes to nested objects or arrays won't affect the original object or vice versa.

**What is currying in javascript?**

Currying is a functional programming concept where a function that takes multiple arguments is transformed into a series of functions that each take a single argument. In other words, when a function is curried, it returns a new function for each argument it's supposed to receive.

### Example:

Consider a simple function that adds two numbers:

**function add(a, b) {**

**return a + b;**

**}**

A curried version of this function would look like:

**function curriedAdd(a) {**

**return function(b) {**

**return a + b;**

**}**

**}**

With the curried version, you can do the following:

**const add5 = curriedAdd(5); // This creates a new function that adds 5 to its input**

**console.log(add5(3)); // Outputs: 8**

There are libraries like Lodash and Ramda that provide utilities for currying functions, making it easier to work with and apply in various scenarios.

**What is difference between arrow function and normal function?**

### this Keyword:

* **Arrow Functions**: They don't have their own **this** context. The value of **this** inside an arrow function is lexically scoped, meaning it's determined by the surrounding code where the arrow function is defined.
* We skip the curly braces and the return keyword. Shorter; one-liner.
* **Normal Functions**: They have their own **this** context, which can change depending on how the function is called (e.g., as a method, standalone function, constructor, etc.).

**Using new keyword:**Regular functions created using function declarations or expressions are ‘constructible’ and ‘callable’. Since regular functions are constructible, they can be called using the ‘new’ keyword. However, the arrow functions are only ‘callable’ and not constructible. Thus, we will get a run-time error on trying to construct a non-constructible arrow function using the new keyword.

The arguments variable does not exist in arrow functions.

A normal function has an arguments object which you can access in the function

In normal functions, a this variable is created which references the objects that call them. For example:

const obj = {

name: 'deeecode',

age: 200,

print: function() {

console.log(this)

}

}

obj.print()

// {

// name: 'deeecode',

// age: 200,

// print: [Function: print]

// }

With normal functions, you can create constructors which serve as a special function for instantiating an object from a class.

Normal functions can be declared when you use the function keyword and a name, but arrow functions cannot be declared. They can only be expressed because they are anonymous:

const printHello = () => {

console.log("hello")

}

As you see here, we have an anonymous function (starting from () => ...) which is assigned to the printHello variable. printHello is not a declared function here. It is a variable that holds the evaluated value from the function expression.

## Arrow functions cannot be accessed before initialization.

**Difference between Jquery Event Listener and React Event Listener:**

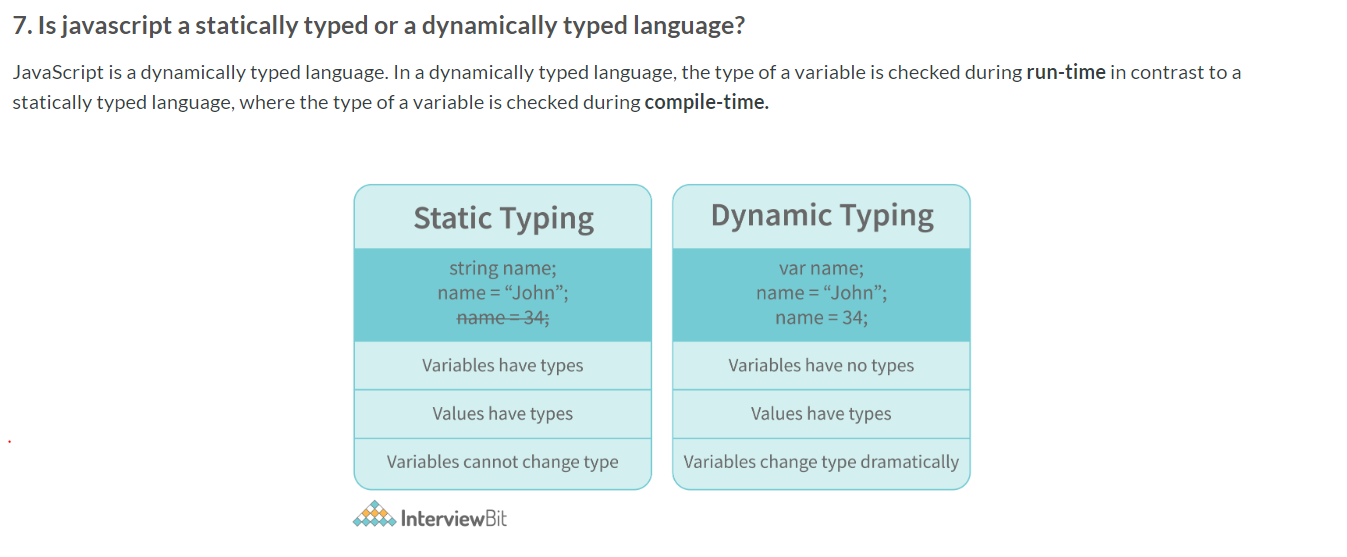
* **React**: Uses a virtual DOM to optimize rendering. When an event changes the state of a component, React batches these updates and uses a diffing algorithm to update the real DOM in the most efficient way possible.
* **jQuery**: Directly manipulates the real DOM, which can be slower for frequent updates or large-scale changes.

**Dependencies and Size**:

* **jQuery**: Requires the inclusion of the jQuery library, which can be relatively large if you're only using it for event handling.
* **React**: While React itself is a larger library, its event system is just a part of its overall architecture. If you're building a React app, you're likely leveraging many other features of the library, making the "cost" of the event system more justified.

**Integration with Component State**:

* **React**: Being a library for building UI components, React tightly integrates event handling with component state. This allows for a more seamless interaction where event handlers can easily update the state of a component, leading to a re-render with the new state.
* **jQuery**: Being primarily a DOM manipulation library, jQuery doesn't have a built-in concept of component state. You'd often have to manually update the DOM in response to events.



### What is an Immediately Invoked Function in JavaScript?

**An Immediately Invoked Function ( known as IIFE and pronounced as IIFY) is a function that runs as soon as it is defined.**

**function**() {

//Do something;

}

// Compiler gives an error since the syntax of declaring a function is wrong in the code above.

To remove this error, we add the first set of parenthesis that tells the compiler that the function is not a function declaration, instead, it’s a function expression.

The second set of parenthesis:

(**function** (){

//Do something;

})();

From the definition of an IIFE, we know that our code should run as soon as it is defined. A function runs only when it is invoked. If we do not invoke the function, the function declaration is returned:

(**function** (){

// Do something;

})// Returns the function declaration

**Therefore to invoke the function, we use the second set of parenthesis.**

### What do you mean by strict mode in javascript and characteristics of javascript strict-mode?

In ECMAScript 5, a new feature called JavaScript Strict Mode allows you to write a code or a function in a "strict" operational environment. In most cases, this language is 'not particularly severe' when it comes to throwing errors. In 'Strict mode,' however, all forms of errors, including silent errors, will be thrown. As a result, debugging becomes a lot simpler.  Thus programmer's chances of making an error are lowered.

Characteristics of strict mode in javascript

1. Duplicate arguments are not allowed by developers.
2. In strict mode, you won't be able to use the JavaScript keyword as a parameter or function name.
3. The 'use strict' keyword is used to define strict mode at the start of the script. Strict mode is supported by all browsers.
4. Engineers will not be allowed to create global variables in 'Strict Mode.

### What do you mean by Self Invoking Functions?

Without being requested, a self-invoking expression is automatically invoked (initiated). If a function expression is followed by (), it will execute automatically. A function declaration cannot be invoked by itself.

Normally, we declare a function and call it, however, anonymous functions may be used to run a function automatically when it is described and will not be called again. And there is no name for these kinds of functions.

**What is the difference between exec () and test () methods in javascript?**

* **test ()** and **exec ()** are RegExp expression methods used in javascript.
* We'll use **exec ()** to search a string for a specific pattern, and if it finds it, it'll return the pattern directly; else, it'll return an 'empty' result.
* We will use a**test ()** to find a string for a specific pattern. It will return the Boolean value 'true' on finding the given text otherwise, it will return 'false'.

### What is currying in JavaScript?

**Currying is an advanced technique to transform a function of arguments n, to n functions of one or fewer arguments.**

Example of a curried function:

**function** **add** (a) {

**return** **function**(b){

**return** a + b;

}

}

add(3)(4)

### Explain Closures in JavaScript.

Closures are an ability of a function to remember the variables and functions that are declared in its outer scope.

**var** Person = **function**(pName){

**var** name = pName;

this.getName = **function**(){

**return** name;

}

}

**var** person = **new** Person("Neelesh");

console.log(person.getName());

**OR ( | | ) operator**- If the first value is truthy, then the first value is returned. Otherwise, always the second value gets returned.

**AND ( && ) operator**- If both the values are truthy, always the second value is returned. If the first value is falsy then the first value is returned or if the second value is falsy then the second value is returned.

**var** x = 220;

**var** y = "Hello";

**var** z = undefined;

x | | y // Returns 220 since the first value is truthy

x | | z // Returns 220 since the first value is truthy

x && y // Returns "Hello" since both the values are truthy

y && z // Returns undefined since the second value is falsy

**if**( x && y ){

console.log("Code runs" ); // This block runs because x && y returns "Hello" (Truthy)

}

**if**( x || z ){

console.log("Code runs"); // This block runs because x || y returns 220(Truthy)

}

### What is memoization?

Memoization is a form of caching where the return value of a function is cached based on its parameters. If the parameter of that function is not changed, the cached version of the function is returned.

**function** **addTo256**(num){

**return** num + 256;

}

addTo256(20); // Returns 276

addTo256(40); // Returns 296

addTo256(20); // Returns 276

### What is recursion in a programming language?

Recursion is a technique to iterate over an operation by having a function call itself repeatedly until it arrives at a result.

function add(number) {

if (number <= 0) {

return 0;

} else {

return number + add(number - 1);

}

}

### What is the use of a constructor function in javascript?

Constructor functions are used to create objects in javascript.

When do we use constructor functions?

If we want to create multiple objects having similar properties and methods, constructor functions are used.

### What do you mean by BOM?

Browser Object Model is known as BOM. It allows users to interact with the browser. A browser's initial object is a window. As a result, you may call all of the window's functions directly or by referencing the window. The document, history, screen, navigator, location, and other attributes are available in the window object

### What are arrow functions?

### Arrow functions are declared without the function keyword. If there is only one returning expression then we don’t need to use the return keyword as well in an arrow function as shown in the example above. Also, for functions having just one line of code, curly braces { } can be omitted.

// Traditional function expression

**var** multiplyBy2 = **function**(num){

**return** num \* 2;

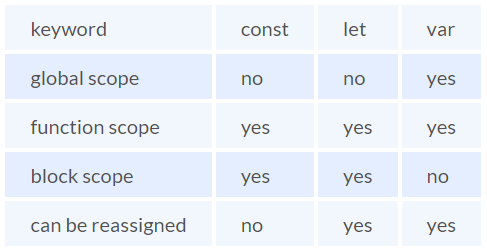
}

// Arrow function expression

**var** arrowMultiplyBy2 = num => num \* 2;

### Differences between declaring variables using var, let and const.

Before the ES6 version of javascript, only the keyword var was used to declare variables. With the ES6 Version, keywords let and const were introduced to declare variables.



* The variables declared with the let keyword in the global scope behave just like the variable declared with the var keyword in the global scope.
* Variables declared in the global scope with var and let keywords can be accessed from anywhere in the code.
* But, there is one difference! Variables that are declared with the var keyword in the global scope are added to the window/global object. Therefore, they can be accessed using window.variableName.  
  Whereas, the variables declared with the let keyword are not added to the global object, therefore, trying to access such variables using window.variableName results in an error.

### What is the rest parameter and spread operator?

Both rest parameter and spread operator were introduced in the ES6 version of javascript.

**Rest parameter ( … ):**

* Using the rest parameter syntax, we can create functions that can take a variable number of arguments.
* **function** **extractingArgs**(...args){
* **return** args[1];
* }
* // extractingArgs(8,9,1); // Returns 9

**In JavaScript, how many different methods can you make an object?**

In JavaScript, there are several ways to declare or construct an object.

1. Object.
2. using Class.
3. create Method.
4. Object Literals.
5. using Function.
6. Object Constructor

### What are generator functions?

Introduced in the ES6 version, generator functions are a special class of functions.  
  
**They can be stopped midway and then continue from where they had stopped.**

**function**\* **genFunc**(){

// Perform operation

}

### Explain WeakSet in javascript.

In javascript, a Set is a collection of unique and ordered elements. Just like Set, WeakSet is also a collection of unique and ordered elements with some key differences:

* Weakset contains only objects and no other type.
* An object inside the weakset is referenced weakly. This means, that if the object inside the weakset does not have a reference, it will be garbage collected.
* Unlike Set, WeakSet only has three methods, **add()**, **delete()**and **has()**.

### Why do we use callbacks?

A callback function is a method that is sent as an input to another function (now let us name this other function "thisFunction"), and it is performed inside the thisFunction after the function has completed execution.

JavaScript is a scripting language that is based on events. Instead of waiting for a reply before continuing, JavaScript will continue to run while monitoring for additional events. Callbacks are a technique of ensuring that a particular code does not run until another code has completed its execution

**What is socket and why we use it in node js ?**

A socket is a fundamental concept in networking that enables communication between two computers over a network. It provides a bi-directional communication channel between a client (such as a web browser or application) and a server (such as a web server or backend application). Sockets allow data to be exchanged in real-time, enabling applications to communicate and share information efficiently.

**Here's why sockets are used in Node.js:**

**Real-Time Communication:** Sockets enable real-time communication between clients and servers. Unlike traditional HTTP requests where the client sends a request and waits for a response, sockets allow continuous two-way communication where both the client and server can send and receive data at any time.

**Event-Driven Architecture:** Node.js is event-driven by nature, and sockets fit well into this model. Sockets use events to handle various stages of communication, such as when a connection is established, data is received, or a connection is closed.

**Bi-Directional Data Flow:** Sockets allow data to flow in both directions simultaneously. This is particularly useful for applications where multiple clients need to be kept up to date with the same information.  
  
  
**What is MondoDB?**

MongoDB is an open-source NoSQL database written in C++ language. It uses JSON-like documents with optional schemas  
MongoDB works on the concept of Collection and Document.  
  
**Document:**

A Document in MongoDB is an ordered set of keys with associated values. It is represented by a map, hash, or dictionary. In JavaScript, documents are represented as objects:  
{"greeting" : "Hello world!"}.  
  
Collection:

A collection in MongoDB is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.  
Documents within a single collection can have any number of different “shapes.”, i.e. collections have dynamic schemas.   
For example, both of the following documents could be stored in a single collection:

{"greeting" : "Hello world!", "views": 3}

{"signoff": "Good bye"}

**Mongo Shell:**

It is a JavaScript shell that allows interaction with a MongoDB instance from the command line. With that one can perform administrative functions, inspecting an instance, or exploring MongoDB.

**Database Indexing**

An index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional storage and some overhead during write operations (like inserts, updates, and deletes).

**Types of Indexes**

* **Primary Index**: Automatically created when a primary key is defined. There is only one primary index per table, and it uniquely identifies each record.
* **Unique Index**: Ensures that the values in the indexed columns are unique. Often used to enforce unique constraints.
* **Clustered Index**: Determines the physical order of data in a table. Each table can have only one clustered index.
* **Non-clustered Index**: Does not alter the physical order of the table and can be multiple per table. It includes a pointer to the data in the form of a row locator.
* **Composite Index**: Uses multiple columns to create an index.
* **Full-text Index**: Specialized index for fast searching of text within large text fields.

**Cluster:**

In Node.js, the cluster module is used to create child processes (workers) that run concurrently and share the same server port. This is particularly useful for taking advantage of multi-core systems, where Node.js itself is single-threaded and can only run on a single core.

the cluster setup described creates multiple worker processes, each of which runs the same server code (i.e., the server.js file). Each worker handles both HTTP and WebSocket connections. All workers are running identical code to handle incoming requests and connections concurrently. Here’s a detailed breakdown to clarify:

**Cluster Master File (cluster.js)**

The cluster master file is responsible for managing the worker processes. It forks multiple workers, each of which runs the server code.

const cluster = require('cluster');

const os = require('os');

if (cluster.isMaster) {

const numCPUs = os.cpus().length;

console.log(`Master ${process.pid} is running`);

// Fork workers

for (let i = 0; i < numCPUs; i++) {

cluster.fork();

}

cluster.on('exit', (worker, code, signal) => {

console.log(`Worker ${worker.process.pid} died`);

// Optionally, you can fork a new worker here

cluster.fork();

});

} else {

require('./server'); // Workers will run the server file

}

**Server File (server.js)**

Each worker process runs the server.js file, handling both HTTP and WebSocket connections.

const http = require('http');

const socketIo = require('socket.io');

// HTTP server setup

const server = http.createServer((req, res) => {

res.writeHead(200);

res.end('Hello World\n');

});

// WebSocket server setup

const io = socketIo(server);

io.on('connection', (socket) => {

console.log('New socket connection:', socket.id);

// Handle socket events here

socket.on('message', (msg) => {

console.log('Message received:', msg);

socket.emit('response', `Received: ${msg}`);

});

socket.on('disconnect', () => {

console.log('Socket disconnected:', socket.id);

});

});

const PORT = 8000;

// Start HTTP server and WebSocket server on the same port

server.listen(PORT, () => {

console.log(`Server listening on port ${PORT}`);

});

### What is MongoDB ?

MongoDB is an open-source NoSQL database written in C++ language. It uses JSON-like documents with optional schemas.

 MongoDB works on the concept of Collection and Document.

### 1. What are some of the advantages of MongoDB?

 MongoDB supports field, range-based, string pattern matching type queries. for searching the data in the database

 MongoDB support primary and secondary index on any fields

 MongoDB basically uses JavaScript objects in place of procedures

 MongoDB uses a dynamic database schema

### 8. What are some features of MongoDB?

* **Indexing:** It supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.
* **Aggregation:** It provides an aggregation framework based on the concept of data processing pipelines.
* **Special collection and index types:** It supports time-to-live (TTL) collections for data that should expire at a certain time
* **File storage:** It supports an easy-to-use protocol for storing large files and file metadata.
* **Sharding:** Sharding is the process of splitting data up across machines.

### 2. Explain the SET Modifier in MongoDB?

If the value of a field does not yet exist, the "$set" sets the value. This can be useful for updating schemas or adding user-defined keys.

Example:

> db.users.findOne()

{

"\_id" : ObjectId("4b253b067525f35f94b60a31"),

"name" : "alice",

"age" : 23,

"sex" : "female",

"location" : "India"

}

To add a field to this, we use “$set”:

> db.users.updateOne({"\_id" :

ObjectId("4b253b067525f35f94b60a31")},

... {"$set" : {"favorite book" : "Start with Why"}})

### 5. Explain the term “Indexing” in MongoDB.

### 1. What do you mean by Transactions?

### 3. What is the Aggregation Framework in MongoDB?

### 5. What is a Replica Set in MongoDB?