const express = require("express");const cors = require("cors");const app = express();app.use(cors()); //cors will be enabled.

useEffect(()=>{ axios.get("http://0.0.0.0:5000/").then((res)=>{ console.log(res); }) axios.post("http://0.0.0.0:5000/addUser",{name:"Saloni", age:16}).then((res)=>{ console.log(res); }) })

<https://corona.lmao.ninja/v2/countries> :api website link

#IFRAME:

<iframe src=”./…….”width=”250”frameborder=”5”>Web Search</iframe>

🡪Now we are using cellpadding, cellspacing, bgcolor, border, rowspan, colspan

<table border=”2”cellpadding=”5”cellspacing=”10”bgcolor=”lightblue”>

<tr rowspan=”2”/”3”> </tr>

<tr colspan=”2”/”3”> </tr>

<tfoot>

<td colspan=”3”>Total:</td>

</tfoot>

🡪for commenting=ctrl+/

#DIV & SPAN

<div> hello </div>

<div>world</div> 🡪hello

World

<span>hello</span>

<span>world</span> 🡪hello world

#SECTION

<section>

<h1>about us</h1>

</section>

<section>

<h1>contact us</h1>

</section>

#HEADER AND NAVIGATION

<header>

Vestige logo

<nav>

<video href="#">Vestige</video>

<a href="#">Dreamers</a>

</nav>

</header>

#FOOTER

<footer>

<div>copyrights</div>

<p>…</p>

</footer>

git config —global user.name = “Your Name”

git config —global user.email = “emailid@gmail.com”

<https://github.com/ravig1729>

<https://css-tricks.com/backdrop-filter-effect-with-css/>

flex-direction : rowmain axis : xcross axis : yflex-direction : columnmain axis : ycross axis : x

justify-content = arrange items on main axisalign-items = arrange items on cross axis

https://github.com/sharma-harshita/operators-in-JS

**Lexical scoping**: Variables declared outside of a function are global variables and are visible everywhere in a JavaScript program. Variables declared inside a function have function scope and are visible only to code that appears inside that function.

Object is a constructor function that refers to all the objects created in the document. We use Object.prototype to add properties or methods to all of the objects that are inheriting from Object

**What are Arrow Functions? Introduced in ES6,it is used to simplify function scoping**.

Arrow functions, introduced in ES6, provides a concise way to write functions in JavaScript. Another significant advantage it offers is the fact that it does not bind its own this. In other words, the context inside arrow functions is lexically or statically defined.

user=[array of elements];

const a=() => {

console.log(user)

}

// Traditional Anonymous Function

function (a, b){

return a + b + 100;

}

// Arrow Function

(a, b) => a + b + 100;

// Traditional Anonymous Function (no arguments)

let a = 4;

let b = 2;

function (){

return a + b + 100;

}

// Arrow Function (no arguments)

let a = 4;

let b = 2;

() => a + b + 100;

**What is DOM?**

HTML is used to **structure**the web pages and Javascript is used to add **behavior**to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM)(Documentary Object Model) **DOM is basically the representation of the same HTML document but in a different format with the use of objects**. Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

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

The constructor () method is a special method for creating and initializing objects created within a class.

Much shorter than using literals every time, and also easy to read, the main purpose of constructors – to implement reusable object creation code. Let’s note once again – technically, any function (except arrow functions, as they don’t have this) can be used as a constructor.

**What is recursion in a programming language?**

Recursion is the process of **repeating items in a self-similar way**. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function, i.e., a function to call itself.

**What is an event loop and call stack?**

It means that the main thread where JavaScript code is run, runs in one line at a time manner and there is no possibility of running code in parallel.

The call stack is used by JavaScript to keep track of multiple function calls. It is like a real stack in data structures where data can be pushed and popped and follows the Last In First Out (LIFO) principle. We use call stack for memorizing which function is running right now.

**What is prototype chain?**

The prototype of an object would also have a prototype object and this continues until we reach the top level when there is no prototype object. This is called prototype chaining.

**What is the use of setTimeout?**

The setTimeout () method in JavaScript is used to execute a function after waiting for the specified time interval. This method returns a numeric value that represents the ID value of the timer. Unlike the setInterval () method, the setTimeout () method executes the function only once.

**What are arrow functions?**

Arrow functions, introduced in ES6, provides a concise way to write functions in JavaScript. Another significant advantage it offers is the fact that it does not bind its own this. In other words, the context inside arrow functions is lexically or statically defined

**Differences between declaring variables using var, let and const**

* var declarations are globally scoped or function scoped while let and const are block scoped.
* var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
* They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
* While var and let can be declared without being initialized, const must be initialized during declaration.

**Explain Closures in JavaScript**.

It is a combination of inner function and the lexical environment created by inner function. We are able to access the variable inside in a function because of closures.

Syntax:

function outer(){

let a =10;

function inner(){

console.log(a);

}inner();

}outer();

**What are object prototypes?**

An object's prototype object may also have a prototype object, which it **inherits methods and properties from, and so on. This is often referred to as a prototype chain**, and explains why different objects have properties and methods defined on other objects available to them.

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");

**What is Object Destructuring?**

JavaScript Object Destructuring is the syntax for extracting values from an object property and assigning them to a variable. The destructuring is also possible for JavaScript Arrays.

By default, the object key name becomes the variable that holds the respective value. So no extra code is required to create another variable for value assignment. Let's see how this works with examples.

const user = {

'name': 'Alex',

'address': '15th Park Avenue',

'age': 43

}

The expression to extract the name property value using object destructuring is the following:

const { name,age } = user;

console.log(name,age); // Output, Alex,43

**What is a Temporal Dead Zone?**

In case of let and const variables, Basically, Temporal Dead Zone is a zone. "before your variable is declared", i.e where you can not access the value of these variables, it will throw an error.

In short, temporal dead zone describes a**zone where variables are un-reachable**. There are variables in the current scope. However, these variables were not declared yet. Try to access those variables inside the temporal dead zone and JavaScript will throw a ReferenceError.

let pi;

let pi =3.14;

what is javascript?

JavaScript is **a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else**.

What is js engine? What are different js engines available?

A JavaScript engine is **a software component that executes JavaScript code**. The first JavaScript engines were mere interpreters, but all relevant modern engines use just-in-time compilation for improved performance. JavaScript engines are typically developed by web browser vendors, and every major browser has one.

Mozilla Firefox: Spider Monkey

Safari: Javascript Core Webkit

Google Chrome: V8.

**What is the for-in loop in JavaScript? Give its syntax**

For-in :returns indexes of the array.

Syntax: const num = [3,4,5,6,7,”Ravi”];

Console.log(“num”);

for(const in N num){

console.log(N)

} //0,1,2,3,4,5

let obj = {

Name: ‘abc’;

}

for (let key in obj){

console.log(obj[key]);

}

**How can one use JavaScript to create an Array?**

As we know, objects allow us to store keyed collections of values. But sometimes we need an ordered collection, where we have to store 1st, 2nd, 3rd element and many more. For instance, consider we need to store a list of something: users’ data, goods data, and many more. It is not appropriate to use an object here because it provides no methods to manage the order of elements. Objects are not created for such use. There exists a [special data structure](https://www.educba.com/data-structure-interview-questions/) called Array to store ordered collections. Arrays in JavaScript enables multiple values to be stored in a single variable. It stores the same kind of element collection sequential fixed-size.

Arrays in JavaScript are used to store an information set, but it is often more helpful for storing a set of variables of the same type. An array is a single variable in JavaScript that is used to store various elements. When we want to store a list of elements and access them through a single variable, it is often used. Unlike many other languages where an array is a various variable reference, there is a single variable in the JavaScript array that stores various elements. The object Arrays in JavaScript are a global object used in array building, which is high-level, list-like objects.

**How to empty an array in JavaScript?**

There are multiple ways to clear/empty an array in JavaScript. You need to use them based on the context. Let us look at each of them. Assume we have an array defined as

let arr = [1, 'test', {}, 123.43];

**Substituting with a new array −**

arr = [];

This is the fastest way. This will set arr to a new array. This is perfect if you don't have any references from other places to the original arr. If you do, those references won't be updated and those places will continue to use the old array.

**Setting length prop to 0 −**

arr.length = 0

This will clear the existing array by setting its length to 0. Fast solution, but this won't free up the objects in this array and may have some memory implications. In order to clean objects in array from memory, they need to be explicitly removed.

**Splice the whole array**

arr.splice(0, arr.length)

This will remove all elements from the array and will actually clean the original array.

**What is creation phase and execution phase?** Creation Phase: In the creation phase, **two unique things** get **created:** A global object called window (for the client-side JavaScript). A global variable called this. If there are any variables declared in the code, the memory gets allocated for the variable. The variable gets initialized with a unique value called undefined.

An execution phase is the**one where the JS engines scan through the function in the code once again**, i.e., one more time for updating the variable object with the values of the variables and then run the code.

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

* Rest Parameters ( … ) Rest parameters help us to pass an infinite number of function arguments. ...
* Spread operator. Spread operator helps us to expand the strings or array literals or object literals. ...Using Spread operator in Function calls. We can use the spread operator to pass an array of numbers as a individual function arguments.

The main difference between rest and spread is that the rest operator puts the rest of some specific user-supplied values into a JavaScript array. But the spread syntax expands iterables into individual elements.

For instance, consider this code that uses rest to enclose some values into an array:

// Use rest to enclose the rest of specific user-supplied values into an array:

function myBio(firstName, lastName, ...otherInfo) {

return otherInfo;

}

// Invoke myBio function while passing five arguments to its parameters:

myBio("Oluwatobi", "Sofela", "CodeSweetly", "Web Developer", "Male");

// The invocation above will return:

["CodeSweetly", "Web Developer", "Male"]

In the snippet above, we used the ...otherInfo rest parameter to put "CodeSweetly", "Web Developer", and "Male" into an array.

Now, consider this example of a spread operator:

// Define a function with three parameters:

function myBio(firstName, lastName, company) {

return `${firstName} ${lastName} runs ${company}`;

}

// Use spread to expand an array’s items into individual arguments:

myBio(...["Oluwatobi", "Sofela", "CodeSweetly"]);

// The invocation above will return:

“Oluwatobi Sofela runs CodeSweetly”

In the snippet above, we used the spread operator (...) to spread ["Oluwatobi", "Sofela", "CodeSweetly"]’s content across myBio()’s parameters.

Let a;

a=[1,2,3,4];

b=a;

b=[…a];

**What is the use of promises in javascript?**

* Promises can handle the asynchronous calls in JavaScript.
* A promise will be "pending" when executed and will result in "resolved" or "rejected", depending on the response of the asynchronous call.
* Promises avoid the problem of "callback hell", which happens due to nested callback functions.

the "[***Callbacks in JavaScript***](https://www.toolsqa.com/javascript/callback-functions-javascript/)" article, callback functions used to handle asynchronous execution. A callback function indicates the operation which JavaScript should execute once an asynchronous operation finishes.

**Explain WeakSet in javascript.**

WeakSet in JavaScript is used to store a collection of objects. It adapts the same properties of that of a set i.e. does not store duplicates. The major difference of a WeakSet with the set is that a WeakSet is a collection of objects and not values of some particular type.

new WeakSet(object)

**Parameters:** Here parameter “object” is an iterable object. All the elements of the iterable object are added to the WeakSet.

var weakSetObject = new WeakSet();

    var objectOne = {};

    var objectTwo = {};

    // add(value)

    weakSetObject.add(objectOne);

    document.write("objectOne added <br>");

    weakSetObject.add(objectTwo);

    document.write("objectTwo added <br>");

    // has(value)

    document.write("WeakSet has objectTwo : " +

                    weakSetObject.has(objectTwo));

    // delete(value)

    weakSetObject.delete(objectTwo);

    document.write("<br>objectTwo deleted<br>");

    document.write("WeakSet has objectTwo : " +

                    weakSetObject.has(objectTwo));

output:

objectOne added

objectTwo added

WeakSet has objectTwo : true

objectTwo deleted

WeakSet has objectTwo : false

**Explain WeakMap in javascript.**

When we want to create a WeakMap, we have to use the **[WeakMap()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakMap/WeakMap)** constructor. This constructor will help us to create a new WeakMap object. We can add new key-value pairs, retrieve, check or remove existing ones.

// Create new WeakMap

const myWeakMap = new WeakMap()

**What is difference between null and undefined and where to use what?**

|  |  |
| --- | --- |
| Null vs Undefined | |
| The null is a data type in JavaScript used to assign a non-value to a variable. | The undefined is a data type in JavaScript used when a variable is declared but not assigned with a value. |
| **Value** | |
| When the variable is assigned to null, the value is null. | When the variable is assigned undefined, the value is undefined. |
| **Data Type** | |
| When the variable is assigned null, the data type is an object. | When the variable is assigned undefined, the data type is considered as undefined. |

When writing programs in JavaScript, it is necessary to store data. Data is stored in variables. Each variable has a data type. Two of them are null and undefined. The  difference between null and undefined in JavaScript is that null is used to assign a non-value to a variable while undefined is used when a variable is declared but not assigned with a value.

Null === null true

Undefined === undefined true

Null === undefined false

**Why do we need callbacks?**

It help us to do achieve the code some later moment. we can map to the eventlisteners. If callbacks are actually "a function statement written inside the argument parenthesis of another function, that then can be called later in the program".

Than, it is unclear to me why we need to use them and just write functions one after the other...

callbacks are necessary for asynchronous programming, including waiting for a user request, making a request to another server and doing something with the response, loading a file, etc. If you used a function synchronously instead of in a callback, then the function would be called before you get the data that the function needs and therefore it wouldn't work. As to the second part of your question, you could declare a function and pass that as the callback instead of passing in an anonymous function, both will work just fine.

function asyncFunction (callback) {

setTimeout(callback, 1000, 'foo'); *// passes foo to the callback*

}

*//This works*

asyncFunction(function(bar) {

console.log(bar)

});

*// this also works*

function bat(bar) {

console.log(bar)

}

asyncFunction(bat)

**What is callback hell?**

In short, callback hell means that you have**multiple functions that are asynchronous**. Those functions depend on each other, which might, in turn, get quite messy with a lot of callback function that is nested in multiple layers. This will result in chaos, and you will end up with code which is hard to read and maintain.

**What is promise chaining?**

The instance method of the [Promise](https://www.javascripttutorial.net/es6/javascript-promises/) object such as then(), catch(), or finally() returns a separate promise object. Therefore, you can call the promise’s instance method on the return Promise. The successively calling methods in this way is referred to as the promise chaining.

Process of executing the asynchronous task in sequential manner.

let p = new Promise((resolve, reject) => {

setTimeout(() => {

console.log(“ “);

resolve(10);

}, 3000);

});

p.then((result) => {

console.log(result);

return result \* 2;

}).then((result) => {

console.log(result);

return result \* 3;

});

**How do you declare strict mode?**

The strict mode is declared by adding "use strict"; to the beginning of a script or a function. If declared at the beginning of a script, it has global scope.

"use strict";

x = 3.14; // This will cause an error because x is not declared

and if you declare inside a function, it has local scope

x = 3.14; // This will not cause an error.

myFunction();

function myFunction() {

"use strict";

y = 3.14; // This will cause an error

}

### What is the difference between window and document?

Below are the main differences between window and document,

| **Window** | **Document** |
| --- | --- |
| It is the root level element in any web page | It is the direct child of the window object. This is also known as Document Object Model(DOM) |
| By default window object is available implicitly in the page | You can access it via window.document or document. |
| It has methods like alert(), confirm() and properties like document, location | It provides methods like getElementById, getElementsByTagName, createElement etc |

### What are the pros and cons of promises over callbacks?

Below are the list of pros and cons of promises over callbacks,

**Pros:**

1. It avoids callback hell which is unreadable
2. Easy to write sequential asynchronous code with .then()
3. Easy to write parallel asynchronous code with Promise.all()
4. Solves some of the common problems of callbacks(call the callback too late, too early, many times and swallow errors/exceptions)

**Cons:**

1. It makes little complex code
2. You need to load a polyfill if ES6 is not supported

### What is the use of setTimeout?

The setTimeout() method is used to call a function or evaluate an expression after a specified number of milliseconds. For example, let's log a message after 2 seconds using setTimeout method,

setTimeout(function(){ console.log("Good morning"); }, 2000);

### What is a prototype chain?

**Prototype chaining** is used to build new types of objects based on existing ones. It is similar to inheritance in a class based language.

The prototype on object instance is available through **Object.getPrototypeOf(object)** or **proto** property whereas prototype on constructors function is available through **Object.prototype**.

[Screenshot](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/prototype_chain.png)

Differences between declaring variables using var, let and const.

* var declarations are globally scoped or function scoped while let and const are block scoped.
* var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
* They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
* While var and let can be declared without being initialized, const must be initialized during declaration.

What are classes in javascript?

class App extends React.Component{  
 render()

return(

)

}

export default App

Day2 :

What is typeOf operator?

Typeof in [JavaScript](https://www.simplilearn.com/reasons-to-learn-javascript-article) is an operator used for type checking and returns the [data type](https://www.simplilearn.com/what-is-data-article) of the operand passed to it. The operand can be any variable, function, or object whose type you want to find out using the typeof operator.

We can use typeof in JavaScript to check the data type of the following operands:

Number, String, Undefined, Boolean, Object, Symbol, Function.

Here, we will pass numbers as operands and use the typeof operator and log the result to the console. We will use a positive integer, negative integer, zero, floating-point number, infinity, NaN, and Math equations as operands. We will also use the concept to explicitly typecasting and parsing a string to an integer or float and use it as an operand. The code below demonstrates the use of all of these operands.

console.log(typeof 12) , console.log(typeof -31), console.log(typeof 0)

**Difference between “==” and “===” operators.**= is called as assignment operator, == is called as **comparison operator whereas It is also called as comparison operator. = does not return true or false**, == Return true only if the two operands are equal while === returns true only if both values and data types are the same for the two variables.

What is the difference between undefine and null ?

In JavaScript, **undefined is a type, whereas null an object**. It means a variable declared, but no value has been assigned a value. Whereas, null in JavaScript is an assignment value. You can assign it to a variable.

What is Nan?

**Not a Number**, is a member of a numeric [data type](https://en.wikipedia.org/wiki/Data_type) that can be interpreted as a [value](https://en.wikipedia.org/wiki/Value_(mathematics)) that is undefined.

What is Functions?

In Javascript, **functions can also be defined as expressions**. For example, // program to find the square of a number // function is declared inside the variable let x = function (num) { return num \* num }; console.log(x(4)); // can be used as variable value for other variables let y = x(3); console.log(y); Run Code.

What is Anonymous Functions?

**Anonymous Function** is a function that does not have any name associated with it. Normally we use the *function*keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the *function*keyword without the function name.

An anonymous function is not accessible after its initial creation, it can only be accessed by a variable it is stored in as a *function as a value*. An anonymous function can also have multiple arguments, but only one expression.

Syntax: var greet = function () {

    console.log("Welcome to GeeksforGeeks!");

};

greet();

What is IIFE?

An Immediately-invoked Function Expression (IIFE for friends) is **a way to execute functions immediately, as soon as they are created**. IIFEs are very useful because they don't pollute the global object, and they are a simple way to isolate variables declarations.

**(function () { //write your js code here })();**

What are High Order Functions?

Higher Orders Functions are functions that perform operations on other functions.In this definition, *operations* can mean taking one or more functions as an argument OR returning a function as the result.

What is meant by first class functions?

A programming language is said to have First-class functions **when functions in that language are treated like any other variable**. For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

function sayHello() {

return "Hello, ";

}

function greeting(helloMessage, name) {

console.log(helloMessage() + name);

}

// Pass `sayHello` as an argument to `greeting` function

greeting(sayHello, "JavaScript!");

// Hello, JavaScript!

What is Function Expressions?

Function Expression **allows us to create an anonymous function which doesn't have any function name** which is the main difference between Function Expression and Function Declaration. A function expression can be used as an IIFE (Immediately Invoked Function Expression)which runs as soon as it is defined. A function expression has to be stored in a variable and can be accessed using *variableName.* With the ES6 features introducing [Arrow Function](https://www.geeksforgeeks.org/arrow-functions-in-javascript/), it becomes more easier to declare function expression.

**DAY 3:**

What is Pure Functions ?

when a same input is passed every time the function will return same output.

function calculate(sum){ 2

return (sum+ 0.5); 2.5

}

calculate(2); //2.5 ; calculate(5); //5.5

A Pure Function is a function (a block of code) that always returns the same result if the same arguments are passed. It does not depend on any state or data change during a program’s execution. Rather, it only depends on its input arguments.

What is Callback Functions ?

callback::

which can be passes as an argument to a function called callback function.

function printA(){

    console.log("A");

}

function printE(){

    console.log("E");

}

printA();     //A

printE();     //E

printA(printE());    //A

What is the generator function?

Generator (or Generator function) is the new concept introduced in ES6. It **provides you a new way of working with iterators and functions**. ES6 generator is a different kind of function that may be paused in the middle either one or many times and can be resumed later.

A generator fun, is **a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values**. In a generator function, a yield statement is used rather than a return statement.

<https://codeburst.io/understanding-generators-in-es6-javascript-with-examples-6728834016d5>

What are operators?

**Operators:** In JavaScript, an operator is **a special symbol used to perform operations on operands (values and variables)**.

1.It will use to compare the value , adding the value.

Airthmatic: +,-,\*,/,%

Comparison Operators : it always returns you the boolean values(true or false) ==, ===, < ,> , <= ,>= ,!=, !==,!= returns when values are not equal , just check the values only.!== return when values are not equal and check the value , datatype.

Increament/Decrement:::a++, ++a, a--, --a,

a++ => a= a+1; variable gets printed first, then its value get incremented.

++a => 1+a ; first value gets incremeented, then var get printed.

a-- => a = a-1;

--a => 1-a;

Logical Operators: && , ||, !.

Mathematical Reasoning:

&& || !

A B A&&B A B A||B A !A

T T T T T T T F

T F F T F T F T

F T F F T T

F F F F F F

Ternary Operators:

<condition> ? <success o/p> : <failure o/p>

10 == 12 ? "present" : "false"

Bitwise OPerators:

LOgic Gates:::::: AND, OR, XOR, NOT, NAND, NOR.

XOR

1 1 0

1 0 1

0 1 1

0 0 0

5 => 0 1 0 1.

&, | ^ , ~, a<<b, a>>b, a>>>b

a<<b => 0 1 0 1 << 1 => 1 0 1 0

a>>b => 0 1 0 1 >> 1 => 0 0 1 0

What is closure in javascript?

Closures: When a fun is running , there is a another variable , we may access the local variable .........It is a combination of inner function and the lexical environment that was created by the inner function.We are able to access the variable inside the function becoz of closures.

function outer(){

let a =10;

function inner() {console.log(a); }

inner();

}outer();

what is function currying. With example?

Currying is **when you break down a function that takes multiple arguments into a series of functions that each take only one argument**. Here's an example in JavaScript: function add (a, b) { return a + b; } add(3, 4); // returns 7. This is a function that takes two arguments, a and b, and returns their sum.

Currying is helpful **when you have to frequently call a function with a fixed argument**. Considering, for example, the following function: If we want to define the function error , warn , and info , for every type, we have two options. Currying provides a shorter, concise, and more readable solution.

Write a function to delete a character from any string?

let str = 'Hello';

str = str.substring(1);

console.log(str);

Define Weakset weakmap?

**Explain WeakSet in javascript.**

WeakSet in JavaScript is used to store a collection of objects. It adapts the same properties of that of a set i.e. does not store duplicates. The major difference of a WeakSet with the set is that a WeakSet is a collection of objects and not values of some particular type.

new WeakSet(object)

**Parameters:** Here parameter “object” is an iterable object. All the elements of the iterable object are added to the WeakSet.

var weakSetObject = new WeakSet();

    var objectOne = {};

    var objectTwo = {};

    // add(value)

    weakSetObject.add(objectOne);

    document.write("objectOne added <br>");

    weakSetObject.add(objectTwo);

    document.write("objectTwo added <br>");

    // has(value)

    document.write("WeakSet has objectTwo : " +

                    weakSetObject.has(objectTwo));

    // delete(value)

    weakSetObject.delete(objectTwo);

    document.write("<br>objectTwo deleted<br>");

    document.write("WeakSet has objectTwo : " +

                    weakSetObject.has(objectTwo));

output:

objectOne added

objectTwo added

WeakSet has objectTwo : true

objectTwo deleted

WeakSet has objectTwo : false

**Explain WeakMap in javascript.**

When we want to create a WeakMap, we have to use the **[WeakMap()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakMap/WeakMap)** constructor. This constructor will help us to create a new WeakMap object. We can add new key-value pairs, retrieve, check or remove existing ones.

// Create new WeakMap

const myWeakMap = new WeakMap()

What is the difference bw Map and forEach?

The main difference between map and forEach is that **the map method returns a new array by applying the callback function on each element of an array, while the forEach method doesn't return anything**,its updates the same array. You can use the forEach method to mutate the source array, but this isn't really the way it's meant to be used.

var salary = [100,200,300,400,500];

console.log('Before Salary',salary);

const newArr = salary.map(x => x+200);

const neweach= salary.forEach((x) =>{if(x > 400){return (x + 10);}});

console.log("After Salary",neweach);

Explain call(), apply() and, bind() methodsExplain Hoisting in javascript

* 1. Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope before code execution.
  2. In JavaScript, a variable can be declared after it has been used.
  3. In other words; a variable can be used before it has been declared.
  4. When we come to functions , It allows us to call functions before even writing them in our code.

1. // Variable lifecycle
2. let a; // Declaration
3. a = 100; // Assignment
4. console.log(a); // Usage

Explain Local Scope, Block Scope, Functional Scope and Scope Chain in javascript

What is meant by first class functions?

First-class Function A programming language is said to have First-class functions **when functions in that language are treated like any other variable**. For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

**DAY 4::**

What is the difference between filter and find?

* 1. .find() will look and stop after the first match, whereas, .filter() will continue searching through the entire array.
  2. [filter](http://api.jquery.com/filter/) reduces the set of already matched elements, while [find](http://api.jquery.com/find/) gets descendants of the matched element.
  3. Both filter() and find() methods are very similar, except the former is applies to all the elements, while latter searches child elements only.

filter() – search through all the elements. find() – search through all the child elements only.

var folks = [

    {name: "Bob", age: "32", occupation: "developer"},

    {name: "Bill", age: "17", occupation: "delinquent"},

    {name: "Brad", age: "40", occupation: "yes"}

  ]

let resu = folks.find( x => x.name === "Bob");

console.log(resu);

//Returns an object: {name: "Bob", age: "32", occupation: "developer"}

let result = folks.filter( y => y.name === "Bob");

console.log(result);

//Returns an array: [ {name: "Bob", age: "32", occupation: "developer"} ]

What is Prototype and Prototype Chaining?

Prototype: An object's prototype ,object may also have a prototype object, which it **inherits methods and properties from, and so on. This is often referred to as a prototype chain**, and explains why different objects have properties and methods defined on other objects available to them.

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");

prototype chain: The prototype of an object would also have a prototype object and this continues until we reach the top level when there is no prototype object. This is called prototype chaining.

What is Call apply bind method?

call(): it is use for calling the fun , we need to give comma separated..,in case of call, each and every arguments which is required to be passed to the function will be passes individually.

displayUserDetails.call(user, "TCS", "developer").

displayUserDetails(user);

Apply() : each and every arguments which is required to be passed to the function will be inside an array.

displayUserDetails.apply(user, ["TCS", "developer"]).

bind(): it will return you new function, and this new function , we can call at anytime and anywhere.

const newFun = displayUserDetails.bind(user);

newFun("Infosys", "Tester")

Difference between Splice and slice.

**The splice() method returns the removed item(s) in an array and slice() method returns the selected element(s) in an array, as a new array object**. The splice() method changes the original array and slice() method doesn't change the original array.

What is Execution Conext?

…

window object v/s document object:

**Document Object:** The document object represent a web page that is loaded in the browser. By accessing the document object, we can access the element in the HTML page. With the help of document objects, we can add dynamic content to our web page. The document object can be accessed with a **window.document**or just**document.**

**Syntax:**

document.property\_name; (body, domain, URL, head, ….)

**Syntax:**

window.method\_name; (createElement, getElementById, …)

**Window Object:** The window object is the topmost object of the DOM hierarchy. It represents a browser window or frame that displays the contents of the webpage. Whenever a window appears on the screen to display the contents of the document, the window object is created.

**Syntax:**

window.property\_name; (console, document…)

**Syntax:**

window.method\_name; (alert, setTimeout,setInterval….)

React :

features of React React v/s angular v/s vue Why you chose react why react is so powerful

JS library developed by Facebook SPA - single page application : Any web application , in which when you are clicking on any button or selecting option from navigation bar then if your page which means browser page is reloading then that means that application is your multi - page application . If it does not reload the browser page and just only updates the page without reloading then that application is known as Single Page application. When you create React application using CRA, (create-react-app boilerplate , developed by Facebook) it always create the application which will be Single page application.

component structure : Whenever we are creating React application so the complete screen component will be broke down into smaller components. We do this thing to make sure that these components can be reused at any time when required. React we have two different types of components : Class and Functional. In React we also create smart and dumb components. Smart are the ones in which state variable is there and you can do all the data manipulation in this. Dumb are the ones which only used for UI, they receive data from parent component as props.

Virtual DOM : DOM is a document object model, created by converting HTML CSS and JS Real DOM, which is an object which gets created whenever any React application gets loaded on the screen for the first time., whenever React components gets mounted on the screen for the first time. Now when any user makes any changes on the screen like button click because of which the state variable will get updated so in this case the changes will not directly go to Real DOM , instead in react we have concept known Virtual DOM. So we are having two virtual doms, one virtual dom gets created at the time of mounting of react component so it is a copy of your real dom. Another virtual dom is the dom which contains the new changes, updated state variables values. Now these two virtual doms will get compared with each other and will check for the new changes. this complete procedure is known as diffing algorithm. Now the new changes will be updated in your Real dom. this procedure is known as Recoinciliation.

Good community support :

Build User interface : used to build Front-end application JSX : Writing HTML inside JS, because in React we write HTML CSS and JS in same file , unlike Angular

Easy learning curve : It requires only the knowledge of Javascript. But in case of Angular it requires Typescript.

React Native : Using React we can create mobile - application as well which supports both Android and iOS.

Client Side rendering one way binding

Library v/s Framework :

Library is a collection of functions : Framework is a collection of multiple other libraries. Inversion of Control = In case of library the Execution of code will be decided by the developers whereas in case of framework the execution is already defined or decided. :In case of Framework, it follows MVC structure (model view controller) whereas a library can only be the one for example React is View of MVC :Framework provides you a structure in which you just put down the code your functionality , in case of library its developers duty to design the structure.

Readme.md : MarkDown Documentation= md This contains information about the project that which file contains what information and also contains info like how a new person who is starting with the project how they have to start with the project

package.json : It contains the information about the project and also contains the libraries which the project is using and it also has the description of the commands which gets used to run test or build your project.

package-lock.json : It contains information about the major libraries and if these libraries are having any extra dependency on other libraries so that information will also be stored over here.

.gitignore: we can list down all the folders or file which we do not want to push during pushing the changes in github like node\_modules.

public/index.html: this file contains one div element in which a id is stored which is used to render your complete application at this id

src/index.js : In this file you will render your complete React application at the id given in index.html file.

src/App.js : From this file ,developers can start building their React application

**Differentiate between Real DOM and Virtual DOM.**

every Real DOM of the document has a Virtual DOM, it is a virtual representation of DOM. That means they are similar to each other according to their structure, elements, and properties but apart from that, they have some difference which based on working of Both DOM as rather than refreshing the DOM when your application state transforms, you just make a Virtual Tree, which seems as though the DOM express that you need. Virtual DOM will at that point make sense of how to make the DOM resemble this effectively without reproducing the entirety of the DOM hubs.

* In Virtual DOM, there is less wastage of memory compare to Real DOM in which too much wastage of memory. This is because Virtual DOM already exists in the memory but in the case of Real DOM, it has to be accessed from the page and loaded to the memory for the operation.
* In Real DOM, manipulation is very expensive because as dealing with the client-side application which was big or dynamic so time spent on elements add up is high whereas In Virtual DOM has less expensive during manipulation due to creating 2 DOM of a different version which helps the developer for adding the elements easily.

**What is React? What are its features**

React is an **open-source front-end JavaScript library** that is used for building user interfaces, especially for single-page applications. It is used for handling view layer for web and mobile apps. React was created by [Jordan Walke](https://github.com/jordwalke), a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

The major features of React are:

* It uses **VirtualDOM** instead of RealDOM considering that RealDOM manipulations are expensive.
* Supports **server-side rendering**.
* Follows **Unidirectional** data flow or data binding.
* Uses **reusable/composable** UI components to develop the view.

Advantages of React over vue and angular:

<https://intellipaat.com/blog/wp-content/uploads/2020/06/Angular-vs-React.jpg>helps the developer for adding the elements easily.

Library vs Framework:

<file:///C:/Users/DELL/Downloads/quicklatex.com-0be15fc7778909b68b7d7feb5a0c49cd_l3.svg>

What is JSX?

*JSX* is a XML-like syntax extension to ECMAScript (the acronym stands for *JavaScript XML*). Basically it just provides syntactic sugar for the React.createElement() function, giving us expressiveness of JavaScript along with HTML like template syntax.

In the example below text inside <h1> tag is returned as JavaScript function to the render function.

class App extends React.Component {

render() {

return(

<div>

<h1>{'Welcome to React world!'}</h1>

</div>

)

}

}

**What do you understand by Virtual DOM? Explain its working?**

The Virtual DOM (VDOM) is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

The Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

**Why can’t browsers read JSX?**

JSX is not a valid JavaScript as they are embedded in HTML elements. As JSX is combination of HTML and JavaScript it is not supported by Browsers. So, if any file contains JSX file, Babel transpiler converts the JSX into JavaScript objects which becomes a valid JavaScript. Thus, browsers understands the code and executes. Browsers can’t read JSX because there is no inherent implementation for the browser engines to read and understand them. JSX is not intended to be implemented by the engines or browsers, it is intended to be used by various transpilers to transform these JSX into valid JavaScript code.

**What is the purpose of render() in React?**

* React renders HTML to the web page by using a function called render().
* The purpose of the function is to display the specified HTML code inside the specified HTML element.
* In the render() method, we can read props and state and return our JSX code to the root component of our app.
* In the render() method, we cannot change the state, and we cannot cause side effects( such as making an HTTP request to the webserver).

**Differentiate between states and props.**

Both props and state are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

**How can you update the state of a component?**

The state is initialized with some value and based on user interaction with the application we update the state of the component at some point in time using**setState method**. setState method allows to change of the state of the component directly using JavaScript object where keys are the name of the state and values are the updated value of that state.

**Explain the lifecycle methods of React components in detail. What are the different phases of React component’s lifecycle?**

## Lifecycle methods:

## Mounting :

When an instance of a component is being created and inserted into the DOM.

## Updating :

When a component is being re-render as a result of changes to either its props or state

## Unmounting :

When a component is being removed from the DOM

## Mounting :

1. constructor
2. static getDerivedStateFromProps
3. render
4. componentDidMount

## Updating :

1. static getDerivedStateFromProps
2. shouldComponentUpdate
3. render
4. getSnapshotBeforeUpdate
5. componentDidUpdate

## Unmounting :

1. componentWillUnmount

//Description:

# Mounting Phase

### Constructor

* A special type of function that will get called whenever a new component is created.
* Used to initialize states & Binding events.
* Not perform, Http req.

### static getDerivedStateFromProps (rarely used)

* When the state of component depends on change in props.
* set the state
* Not perform, Http req.

### render

* Only Required Method
* return JSX
* Children component Lifecycly methods also get execute
* Not perform, Http req.

### componentDidMount

* Invoked immediately after a component and its child components have been rendered to DOM
* Perform any AJAX call to load your data.

### Updating Phase

## static getDerivedStateFromProps

* Method is called every time a component is re-rendered.
* set the state
* Not perform, Http req.

## shouldComponentUpdate

* Dictates if the component should re-render or not.
* Perfomance Optimization
* Not perform, Http req.

## Render

* Only Required Method
* Return JSX
* Not perform, Http req.

## getSnapshotBeforeUpdate() [Rarely Used]

* Called right before the changes from the virtual DOM are to be reflected in the DOM.
* Capture some Information From DOM

## componentDidUpdate()

* called after the render is finished in the re-render Cycles.

### Unmounting Phase ( LAST WISH )

* Method is invoked immediately before a component is unmounted and destroyed.
* Cancelling any network req. , also invalidating timers.
* DO NOT CALL THE SetSTATE METHOD

**Differentiate between stateful and stateless components.**

**Stateless:**

If the behaviour is independent of its state then it can be a stateless component. You can use either a function or a class for creating stateless components. But unless we need to use a lifecycle hook in your components, we should go for function components. There are a lot of benefits if you decide to use function components here; they are easy to write, understand, and test, a little faster, and you can avoid the this keyword altogether.

Stateful:

If the behaviour of a component is dependent on the *state* of the component then it can be termed as stateful component. These *stateful components* are always *class components* and have a state that gets initialized in the constructor.

class App extends Component {

constructor(props) {

super(props)

this.state = { count: 0 }

}

render() {

// ...

}

}

**React 16.8 Update:**

Hooks let you use state and other React features without writing classes.

*The Equivalent Functional Component*

import React, {useState} from 'react';

const App = (props) => {

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

return (

// JSX

)

}

**What are synthetic events in React?**

SyntheticEvent is a cross-browser wrapper around the browser's native event. Its API is same as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.

How do you conditionally render components?

In some cases you want to render different components depending on some state. JSX does not render false or undefined, so you can use conditional *short-circuiting* to render a given part of your component only if a certain condition is true.

const MyComponent = ({ name, address }) => (

<div>

<h2>{name}</h2>

{address &&

<p>{address}</p>

}

</div>

)

If you need an if-else condition then use *ternary operator*.

const MyComponent = ({ name, address }) => (

<div>

<h2>{name}</h2>

{address

? <p>{address}</p>

: <p>{'Address is not available'}</p>

}

</div>

)

What are the differences between functional and class components?

|  |  |
| --- | --- |
| **Functional Components** | **Class Components** |
| A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element. | A class component requires you to extend from React. Component and create a render function which returns a React element. |
| There is no render method used in functional components. | It must have the render() method returning JSX (which is syntactically similar to HTML) |
| Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI. | Also known as Stateful components because they implement logic and state. |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components. | React lifecycle methods can be used inside class components (for example, componentDidMount). |
| Hooks can be easily used in functional components to make them Stateful.  example: const [name,SetName]= React.useState(‘ ‘) | It requires different syntax inside a class component to implement hooks.  example: constructor(props) {     super(props);     this.state = {name: ‘ ‘}  } |
| Constructors are not used. | Constructor are used as it needs to store state. |

How to prevent re-renders in React?

* Don’t change the state in the main body of the component.
* Use the useEffect hook very cautiously. The second parameter of useEffect is an array of states based on the useEffect will call. So don’t update those states in useEffect otherwise, it will rerender the component again and again.
* **Use shouldComponentUpdate** is a method for optimizing performance, which tells React to stop re-rendering a component, even though it might have changed the state or prop values. Using this approach only if a part stays unchanged or pure while it is used. You are expected to return a Boolean value with the React shouldComponentUpdate method. Return true if it needs to re-render or false to avoid being re-render.

What is the difference between Element and Component?

An *Element* is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. *Elements* can contain other *Elements* in their props. Creating a React element is cheap. Once an element is created, it is never mutated.

The object representation of React Element would be as follows:

const element = React.createElement(

'div',

{id: 'login-btn'},

'Login'

)

The above React.createElement() function returns an object:

{

type: 'div',

props: {

children: 'Login',

id: 'login-btn'

}

}

And finally it renders to the DOM using ReactDOM.render():

<div id='login-btn'>Login</div>

Whereas a **component** can be declared in several different ways. It can be a class with a render() method or it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output:

const Button = ({ onLogin }) =>

<div id={'login-btn'} onClick={onLogin}>Login</div>

Then JSX gets transpiled to a React.createElement() function tree:

const Button = ({ onLogin }) => React.createElement(

'div',

{ id: 'login-btn', onClick: onLogin },

'Login'

)

What is CRA and its benefits?

The create-react-app CLI tool allows you to quickly create & run React applications with no configuration step.

Let's create Todo App using *CRA*:

# Installation

$ npm install -g create-react-app

# Create new project

$ create-react-app todo-app

$ cd todo-app

# Build, test and run

$ npm run build

$ npm run test

$ npm start

It includes everything we need to build a React app:

1. React, JSX, ES6, and Flow syntax support.
2. Language extras beyond ES6 like the object spread operator.
3. Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
4. A fast interactive unit test runner with built-in support for coverage reporting.
5. A live development server that warns about common mistakes.
6. A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

Can you force a component to re-render without calling setState?

By default, when your component's state or props change, your component will re-render. If your render() method depends on some other data, you can tell React that the component needs re-rendering by calling forceUpdate().

component.forceUpdate(callback)

It is recommended to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

What is the difference between super() and super(props) in React using ES6 classes?

When you want to access this.props in constructor() then you should pass props to super() method.

**Using super(props):**

class MyComponent extends React.Component {

constructor(props) {

super(props)

console.log(this.props) // { name: 'John', ... }

}

}

**Using super():**

class MyComponent extends React.Component {

constructor(props) {

super()

console.log(this.props) // undefined

}

}

Outside constructor() both will display same value for this.props.

What are fragments? Why fragments are better than container divs?

Fragments: It's a common pattern in React which is used for a component to return multiple elements. *Fragments* let you group a list of children without adding extra nodes to the DOM.

render() {

return (

<React.Fragment>

<ChildA />

<ChildB />

<ChildC />

</React.Fragment>

)

}

There is also a *shorter syntax*, but it's not supported in many tools:

render() {

return (

<>

<ChildA />

<ChildB />

<ChildC />

</>

)

}

Below are the list of reasons for fragments are better than container divs:

1. Fragments are a bit faster and use less memory by not creating an extra DOM node. This only has a real benefit on very large and deep trees.
2. Some CSS mechanisms like *Flexbox* and *CSS Grid* have a special parent-child relationships, and adding divs in the middle makes it hard to keep the desired layout.
3. The DOM Inspector is less cluttered.

**What are Higher Order Components(HOC)?**

A *higher-order component* (*HOC*) is a function that takes a component and returns a new component. Basically, it's a pattern that is derived from React's compositional nature.

We call them **pure components** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

const EnhancedComponent = higherOrderComponent(WrappedComponent)

Why do you need HOC?

HOC can be used for many use cases:

1. Code reuse, logic and bootstrap abstraction.
2. Render hijacking.
3. State abstraction and manipulation.
4. Props manipulation.

**What are Pure Components?**

*React.PureComponent* is exactly the same as *React.Component* except that it handles the shouldComponentUpdate() method for you. When props or state changes, PureComponent will do a shallow comparison on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

**Explain React Hooks.**

Hooks are the new feature introduced in the **React** 16.8 version. It allows you to use state and other **React** features without writing a class. Hooks are the functions which "**hook** into" **React** state and lifecycle features from function components. It **does** not work inside classes.

Before React Hooks (React < 16.8), developer's were required to write class components in order to take advantage of certain React Features. But now, React Hooks provides a more ergonomic way to build components because we can use stateful logic without changing our component hierarchy.

You need to follow two rules in order to use hooks,

1. Call Hooks only at the top level of your react functions. i.e, You shouldn’t call Hooks inside loops, conditions, or nested functions. This will ensure that Hooks are called in the same order each time a component renders and it preserves the state of Hooks between multiple useState and useEffect calls.
2. Call Hooks from React Functions only. i.e, You shouldn’t call Hooks from regular JavaScript functions.

**Explain useState, useEffect, useContext, useReducer, useRef.**

### useState :It is the most important and often used hook. The purpose of this hook to handle reactive data, any data that changes in the application is called state, when any of the data changes, React re-renders the UI.

const [count, setCount] = React.useState(0);

### 🚀 useEffect :It allows us to implement all of the lifecycle hooks from within a single function API.

// this will run when the component mounts and anytime the stateful data changes

React.useEffect(() => {

alert('Hey, Nads here!');

});

// this will run, when the component is first initialized

React.useEffect(() => {

alert('Hey, Nads here!');

}, []);

// this will run only when count state changes

React.useEffect(() => {

fetch('nads').then(() => setLoaded(true));

}, [count]);

// this will run when the component is destroyed or before the component is removed from UI.

React.useEffect(() => {

alert('Hey, Nads here');

return () => alert('Goodbye Component');

});

### 🚀 useContext :This hook allows us to work with React's Context API, which itself a mechanism to allow us to share data within it's component tree without passing through props. It basically removes prop-drilling

const ans = {

right: '✅',

wrong: '❌'

}

const AnsContext = createContext(ans);

function Exam(props) {

return (

// Any child component inside this component can access the value which is sent.

<AnsContext.Provider value={ans.right}>

<RightAns />

</AnsContext.Provider>

)

}

function RightAns() {

// it consumes value from the nearest parent provider.

const ans = React.useContext(AnsContext);

return <p>{ans}</p>

// previously we were required to wrap up inside the AnsContext.Consumer

// but this useContext hook, get rids that.

}

### useReducer :

It does very similiar to setState, It's a different way to manage state using Redux Pattern. Instead of updating the state directly, we dispatch actions, that go to a reducer function, and this function figure out, how to compute the next state.

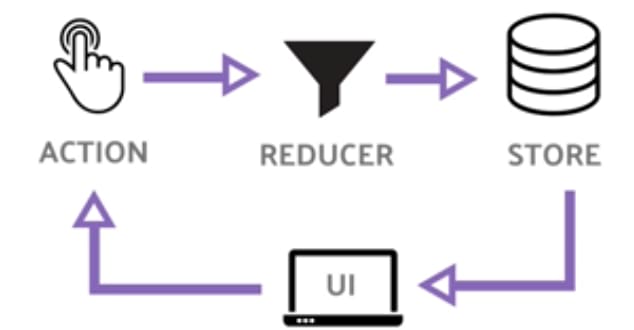
[](https://res.cloudinary.com/practicaldev/image/fetch/s--ik_MbBeK--/c_limit,f_auto,fl_progressive,q_auto,w_880/https:/dev-to-uploads.s3.amazonaws.com/i/885ulztfmsm2xhdqj3mp.png)

Fig. useReducer Architecture

function reducer(state, dispatch) {

switch(action.type) {

case 'increment':

return state+1;

case 'decrement':

return state-1;

default:

throw new Error();

}

}

function useReducer() {

// state is the state we want to show in the UI.

const [state, dispatch] = React.useReducer(reducer, 0);

return (

<>

Count : {state}

<button onClick={() => dispatch({type:'decrement'})}>-</button>

<button onClick={() => dispatch({type:'increment'})}>+</button>

</>

)

}

**useRef :** is useful whenever you want to reference the same value on every render, but not trigger a re-render when its value changes.

useRef returns a mutable ref object whose .current property is initialized to the passed argument (initialValue). The returned object will persist for the full lifetime of the component.

**What do you know about controlled and uncontrolled components?**

Controlled: A component that controls the input elements within the forms on subsequent user input is called **Controlled Component**, i.e, every state mutation will have an ascontextsociated handler function.

For example, to write all the names in uppercase letters, we use handleChange as below,

handleChange(event) {

this.setState({value: event.target.value.toUpperCase()})

}

Uncontrolled: The alternative is uncontrolled components, where form data is handled by the DOM itself.

The **Uncontrolled Components** are the ones that store their own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

In the below UserProfile component, the name input is accessed using ref.

class UserProfile extends React.Component {

constructor(props) {

super(props)

this.handleSubmit = this.handleSubmit.bind(this)

this.input = React.createRef()

}

handleSubmit(event) {

alert('A name was submitted: ' + this.input.current.value)

event.preventDefault()

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

{'Name:'}

<input type="text" ref={this.input} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

**What is React Router?**

React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.

**Why do we need a Router in React?**

Need of React Router. React Router plays an important role to **display multiple views in a single page application**. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

Difference Between Browser Router as Router, Routes and Route.

**BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the URL**. It is the parent component that is used to store all of the other components.

Difference between Link and Navlink.

the main difference between these two’s is a class attribute. When we use the NavLink as a tag, it automatically inherit an active class when clicked. On the other hand, the Link tag does now have an active class when clicked.

Implement Browser Router and create a navigation bar………………….

**What is Redux?**

Redux is a predictable state container for JavaScript apps based on the Flux design pattern. Redux can be used together with React, or with any other view library. It is tiny (about 2kB) and has no dependencies.

**What are the three principles that Redux follows?**

1. **Single source of truth:** The state of your whole application is stored in an object tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
2. **State is read-only:** The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state.
3. **Changes are made with pure functions:** To specify how the state tree is transformed by actions, you write reducers. Reducers are just pure functions that take the previous state and an action as parameters, and return the next state.

List down the components of Redux. and explain.

The components of Redux architecture are explained below.

**STORE:** A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

**ACTION:** Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

**REDUCER:** Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

Client Side Rendering v/s Server Side Rendering.

Registeration/Login Process:::::::::::

Registeration:::

1. FE will display the input fields that you want to capture from the user. Name, age, salary, dept, email, password etc.

2. User will be providing the details in the fields, then he will click on submit button or register/ signup button.

3. FE will verify if all the fields are properly filled and correct. -Validating input fields: Regex regex101.com (Form Validation)

4. FE will be having an API call which will pass this complete data in API to the backend.

5. Validate email is already present or not. Now its time to store the details in DB. ENCRYPTION :Information like passwords should not be stored in the db directly. So we will encrypt the password and then only store it. So this encryption code will be written in controllers

node.bcyript.js

6. There is a library known as bcrypt npm any text will be converted into a hash value

7. Once the bcrypt has returned you hashed value, then we will store that in DB along with other user details.

**Login:**

1. FE will take data from user email/username and password

2. Click on login button, then FE will call an API from backend and will pass email and password to it.

3. APIS comes up with different types of error messages, like if email and password is not provided then msg would be "Please provided required details", if email is not already present in database, then msg would be : "Given email is not registered, Please register". Password's hashed value will be created once again and then it will compare it with the one saved in DB. if it returns then home page if not then error msg will be "Your password is incorrect, please try with correct password."

4.BE/login api will provide the token to the frontend. This tokens will be generated using library name- jsonwebtoken.

5.After receiving this token from BE, FE will store this token in the local storage. and the whatever next api calls has to be made, this token will be passed in the Headers. inside authorization field concatenated with Bearer and then token.

6.IRCTC Applications: Token Expiration

/login-it will give you token , it will say that it should expire in 5 mins.

/contact –after 4 mins. - this api user is trying to access by passing token, at this time he will get the response as well from BE.

/user – after 5 mins – if the token will be expired by now, so the error will be sent to FE, noe FE responsibility would be to send the user back to login page is such error occurs. Error msgs would be: Session has expired , please re-login.

Authentication – form validation/user details are correct or not …

and Authorisation::::role based access.

**Authentication is used to authenticate someone's identity, whereas authorization is a way to provide permission to someone to access a particular resource.**

authentication validates that users are who they claim to be. Usually with a username and password, (401 status code for unAuthentication). Authorization checks, whether that user has permission to access aspecific resource once they've been authenticated sometimes called access control because it controls which resources in the server the current user has access to. (403 status code forbidden)

**NODEJS**

What is NPM?

npm (Node Package Manager) is a package manager for the JavaScript programming language maintained by npm, Inc. npm is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry. The registry is accessed via the client, and the available packages can be browsed and searched via the npm website.

What are the modules in Node.js?

Consider modules to be the same as JavaScript libraries. A set of functions you want to include in your application.

Node.js has a set of built-in modules which you can use without any further installation like buffer(to handle binary data),cluster(to split single Node process into multiple processes),events(to handle events),fs(to handle the file system), http and https(to make nodejs acts as an HTTP AND HTTPS server), timers( to execute a function after a given number of milliseconds), v8(To access information about V8).

To include a module, use the require() function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server

What is the purpose of the module .Exports?

The main purpose of module.exports is**to achieve modular programming**. Modular programming refers to separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Difference between default export and named export

ES6 provides us to import a module and use it in other files. Strictly speaking in React terms, one can use stateless components in other components by exporting the components from their respective modules and using it in other files.ES6 provides two ways to export a module from a file: named export and default export.

**Named Export: (export)**

With named exports, one can have multiple named exports per file. Then import the specific exports they want surrounded in braces. The name of imported module has to be the same as the name of the exported module.

// imports  
// ex. importing a single named export  
import { MyComponent } from "./MyComponent";// ex. importing multiple named exports  
import { MyComponent, MyComponent2 } from "./MyComponent";// ex. giving a named import a different name by using "as":  
import { MyComponent2 as MyNewComponent } from "./MyComponent";// exports from ./MyComponent.js file  
export const MyComponent = () => {}  
export const MyComponent2 = () => {}

Import all the named exports onto an object:

import \* as MainComponents from "./MyComponent";  
// use MainComponents.MyComponent and MainComponents.MyComponent2  
here

**Default Export: (export default)**

One can have only one default export per file. When we import we have to specify a name and import like:

// importimport MyDefaultComponent from "./MyDefaultExport";// exportconst MyComponent = () => {}export default MyComponent;

## How do you import any module in Node.js?

The require () function is used to import modules in node.js.

What are the different types of HTTP requests?

* GET is used to retrieve and request data from a specified resource in a server. GET is one of the most popular HTTP request techniques.
* HEAD: The HEAD technique requests a reaction that is similar to that of GET request, but doesn’t have a message-body in the response.
* POST. In web communication, POST requests are utilized to send data to a server to create or update a resource.
* PUT. is similar to POST as it is used to send data to the server to create or update a resource. ...
* DELETE. request method is used to delete resources indicated by a specific URL.

Explain the concept of middleware in Node.js.

**Middleware** functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named next.

1. As name suggests it comes in middle of something and that is request and response cycle
2. Middleware has access to **request**and **response**object
3. Middleware has access to **next**function of request-response life cycle

Explain CORS.

Cross-Origin Resource Sharing (CORS) is the mechanism to allow content loaded from one (main) origin to access the selected resources available from servers at different origin. This mechanism of accessing resources at different origin is called CORS (Cross-Origin Resource Sharing). This is possible by making use of additional HTTP headers by both browser and Cross Origin server.

CORS is the mechanism among others to relax the browsers [**Same-Origin Policy**](https://en.wikipedia.org/wiki/Same-origin_policy)

What is Express. how it helps you to create a backend application?