1. What is redux. Why we use in react.
2. What is uni-directional data flow.
3. Different between left join and left inner join
4. What is server side rendering
5. Why we are not mongo db in large projects
6. What is collections in mongo db
7. What is different between node js react js
8. What is react life cycle
9. Which life cycle is execute both server side and client side both.
10. How to compare two objects? \_.isEqual(objct1,object2);   
    **in cognizant**
11. **What is spread operator**
12. **What is closures in javascript**
13. **What is arrow operator and how to bind context in this**
14. **What is store in redux.**
15. **What is features of ES6..**
16. **What is promise?**
17. **What is middleware ?**
18. **What is mutable and immutable (state and props) ?**
19. **What is difference bt foreach and map loop in react**
20. **What is difference bt mongodb and moogoose.**
21. **About filters.**
22. **What is difference bt indexof and filters**
23. **Difference bt react.createElement and react.createDom**
24. **HOC**
25. **How to define real component ?**
26. **How to create pure component.**
27. **mutable and immutable objects**
28. **d structuring**
29. **hookes in react js**
30. **event sourcing in react js**
31. **programming pattern**
32. **filling and drafting**
33. **apply and call**
34. **What is redux. Why we use in react.**
35. **function expresion and function defination**
36. **how to handle auto complete in html 5**
37. **how to connect the container in redux**
38. **what is context arrow funnction**
39. **[1,1,1,1,2,2,2,2,3,3,3]**
40. **error handling in react js**
41. **error handling life cycle in react**
42. **authentication in react**
43. **testing in react**
44. **conditional routing in react js**
45. **split and join**
46. **let, var, constant in js**
47. **what is different bt function xyz(), and const xyz=()=>{}**
48. **how many types of defining a function**
49. **hosting in js**
50. **event bubling and event sourcing**
51. **what is bind**
52. **What is uni-directional data flow**

It means that data flows throughout your app in a single direction. and you have better control over it (we'll see how). Also the application state is contained in specific stores and as a result different components of your app remain loosely coupled.

1. **What is spread operator**

spread (...) operator to copy enumerable properties from one object to another in a more succinct way.  
function todoApp(state = initialState, action) {

switch (action.type) {

case SET\_VISIBILITY\_FILTER:

return { ...state, visibilityFilter: action.filter }

default:

return state

}

}

var colors = ['red', 'green', 'blue'];  
var refColors = [...colors, 'yellow'];

1. **What is closures in javascript**a closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

function showName (firstName, lastName) {  
function makeFullName () {    
return nameIntro + firstName + " " + lastName;    
}

return makeFullName ();

}

1. **What is ES6 ?**This is a version of ECMA sript.In 1996 Netscape submitted JavaScript to ECMA International for standardization. EcmaScript is the standardized scripting language that JavaScript (and some other languages, like ActionScript) implement

Features:

ES6 introduces language support for classes (class keyword), constructors (constructor keyword), and the extend keyword for inheritance.

\* Arrow function

\*require to import and export

**\* What is Promises ?**

**Promises**

Promises provide a mechanism to handle the results and errors from asynchronous operations. You can accomplish the same thing with callbacks, but promises provide improved readability via method chaining and succinct error handling.

A promise is used to handle the asynchronous result of an operation. JavaScript is designed to not wait for an asynchrnous block of code to completely execute before other synchronous parts of the code can run.

Promises have three states:

Pending: This is the initial state of the Promise before an operation begins

Fulfilled: This means the specified operation was completed

Rejected: The operation did not complete; an error value is usually thrown

**Using Promises**

Using a promise that has been created is relatively straightforward; we chain .**then()** and .**catch**() to our Promise like so:

* **What is Callbacks function**

When a function simply accepts another function as an argument, this contained function is known as a callback function.

3.**What is arrow operator and how to bind context in this**  
 It has reduced the function defining code size   
 setDoubleSize = (sz)=>size=2\*sz;

**4. What is mutable and immutable (state and props) ?**

**Props** are used to pass data from parent to child or by the component itself. They are immutable and thus will not be changed.

**State** is used for mutable data, or data that will change.

1. **What is difference bt foreach and map loop in react**

forEach() — executes a provided function once for each array element.but not return any thing in new variable

map() — creates a new array with the results of calling a provided function on every element in the calling array.

let arr = [1, 2, 3, 4, 5];

**arr.forEach((num, index) => {**

**return arr[index] = num \* 2;**

**});** /// // arr = [2, 4, 6, 8, 10]

**let doubled = arr.map(num => {**

**return num \* 2;**

**});**

// // doubled = [2, 4, 6, 8, 10]

1. **What is difference bt mongodb and moogoose.**

**Mongodb**:

It's a database.

**Mongoose**:

It's an object modeling tool.

1. **Why map, filter, reduce?**

In Javascript we have map, filter and reduce, all functions that given an initial list (array of things), transform it into something else, while keeping that same original list intact.

**Filter** receives the same arguments as **Map**, and works very similarly. The only difference is that the callback needs to return either true or false. If it returns true then the array keeps that element and if it returns false the element is filtered out.

Last but not least, **reduce** takes an array and reduces it into a single value.

**var sample = [1, 2, 3];**

**FOREACH**

sample.forEach((elem, index) => `${elem} comes at ${index}`)

/\*

output

1 comes at 0

2 comes at 1

3 comes at 2

\*/

**Filter**

var result = sample.filter(elem => elem !== 2) // [1, 3]

**Map**

makes it unique is it generate a new array based on your existing array.

let mapped = sample.map(elem => elem \* 10) // [10, 20, 30]

**Reduce**

As the name already suggest reduce method of the array object is used to reduce the array to one single value.

var sum = sample.reduce((sum, elem) => sum + elem) //

1. What is different between **forEach and filter**

The main difference between forEach and filter is that forEach just loop over the array and executes the callback but filter executes the callback and check its return value.

1. **What is include in javascript.**

The includes() method determines whether a string contains the characters of a specified string. This method returns true if the string contains the characters, and false if not.The includes() method is case sensitive.

1. **What is different between IndexOf and search.**

**IndexOf()** - it accepts string literals or string objects but not regular expressions. It also accepts a zero-based integer value to start its search from

"babyelephant".indexOf("e"); // gives you 4

"babyelephant".indexOf("e",5); // gives you 6

**Search()** - accepts both string literals or string objects and regular expressions. But it doesn't accepts a index to start the search from

"hello.".search(".") // 0 not 5 because . is the regex token for "any character"

1. **nd immutable objects in javascript** ?

**Mutable object** - The object is subject to be changed/altered.  
**Immutable object** - The object cannot be changed once created.

**15 What is destructuring assignment**The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

var a, b, rest;  
[a, b] = [10, 20];  
console.log(a); // expected output: 10  
console.log(b); // expected output: 20  
[a, b, ...rest] = [10, 20, 30, 40, 50];  
console.log(rest); // expected output: [30,40,50]

1. **Hooks at a Glance**

Hooks are an upcoming feature that lets you use state and other React features without writing a class.

* **event sourcing in react js**

**Event Sourcing** is a software architecture pattern which makes it possible to reconstruct past states (latest state as well). It's achieved in a way that every state change gets stored as a sequence of events.

**There's a very strong analogy with Redux here, I think. Actions are events, and using pure reducer functions to change state guarantees that replaying the actions will get you back the same state.**

* **What is programming pattern ?**

it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

* **filling and drafting**

A design doc — also known as a technical spec — is a description of how you plan to solve a problem.

* **apply and call In javascript**.

Call take same parameter as main functions is declared

Apply take second parameter in array form.

Function abc (name,roll)

{

Return name;

}

Abc.call(undefined,’vinay’,’2’);

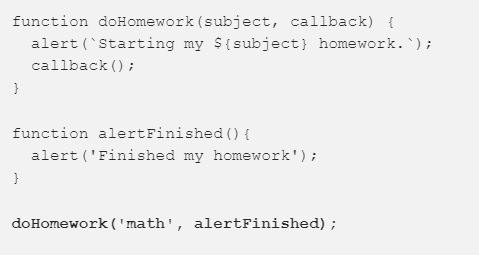
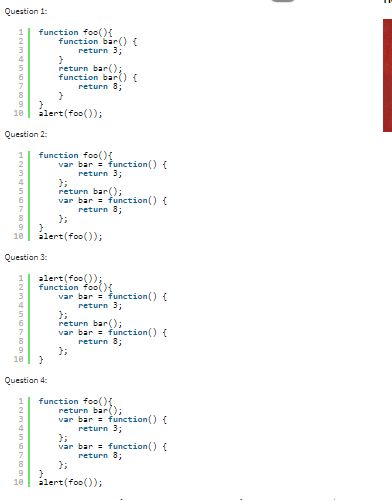
Abc.apply(undefined,[‘vinay’,’2’]);

* **slice()**

The slice() method returns the selected elements in an array, as a new array object.

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];

var citrus = fruits.slice(1, 3); // Orange,Lemon

* **what is callback in javascript**  
  A callback is a function that is to be executed after another function has finished executing — hence the name ‘call back’.  
  
* **What is callback and higher-order-function**A functions that can take functions as arguments, and can be returned by other functions. Functions that do this are called **higher-order functions**. Any function that is passed as an argument is called a callback function.
* **Why do we need Callbacks?**JavaScript is an event driven language. This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events.
* **What is callback hell.**The situation where callbacks are nested within other callbacks several levels deep, potentially making it difficult to understand and maintain the code
* **What is wait for**wait.for is using for the handle the promises.
* **What is function declaration and function expression  
  **

Answers : 8, 3, 3 and Type Error

**Function Declaration?**

A Function Declaration defines a named function variable without requiring variable assignment.

function bar() {

return 3;

}

**Function Expression?**

A JavaScript function can also be defined using an expression.A function expression can be stored in a variable

var x = function (a, b) {return a \* b};

* **What is function declaration and function expression**A Function **Expression** is created when the execution reaches it and is usable from then on.

A Function **Declaration** is usable in the whole script/code block.  
Function **expressions** aren’t hoisted, which allows them to retain a copy of the local variables from the scope where they were defined. But **declarations** are hoisted

* **what is hoisting?**Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).Basically, when Javascript compiles all of your code, all variable declarations using var are hoisted/lifted to the top of their functional/local scope (if declared inside a function) or to the top of their global scope (if declared outside of a function) regardless of where the actual declaration has been made. This is what we mean by “hoisting”.Functions declarations are also hoisted, but these go to the very top, so will sit above all of the variable declarations.
* **Var , let and const in javascript**Variables and constants declared with let or const are not hoisted! But var is hoisted

**for node js**

1) what is call back

2) call back hell

3) node js multi threaded or single threaderd

4) srading in node js

5) how to create index in mongo db

6) how to use authentication

cello cpy deep copy

arrow function has there own context

what is props.children

functiona programming and pure function

updated life cycle events

setTimeOut 0 mini

second

window rwady and window load

call back hell

event loop \*\*\* in node js

ascy, sync and non blocking

* **REPL**A read–eval–print loop (REPL) is a simple, interactive computer programming environment.In short, it starts an interactive console where you can type in commands and immediately see the result of these commands.
* **View state and session state**
* **Access modifires**
* **Data abstrictoin**
* **Inheritane and data abstraction**
* **What is first parameter of call back function**function(err, results){}
* **What is functional programming ?**

**Callback functions** are derived from a programming paradigm known as **functional programming**. At a fundamental level, functional programming specifies the use of functions as arguments. Functional programming was—and still is, though to a much lesser extent today—seen as an esoteric technique of specially trained, master programmers.

* How to get interests value of two array In js.  
  const as = new Set([1, 2, 3]);  
  const bs = new Set([3, 2, 4]);  
  const intersection = [...as].filter(bs.has, bs);  
   // [2, 3]

**64. Define event bubbling?**

<form onclick="alert('form')">FORM

<div onclick="alert('div')">DIV

<p onclick="alert('p')">P</p>

</div>

</form

P : p, div, form

Div : div, form

Form : form

* **Example of apply and call in javascript**

<var someObject = {

myProperty : 'Foo',

myMethod : function(prefix, postfix) {

alert(prefix + this.myProperty + postfix);

}

};

/\* someObject.myMethod('<', '>'); // alerts '<Foo>' \*/

var someOtherObject = {

myProperty : 'Bar'

};

someObject.myMethod.call(someOtherObject, '<', '>'); // alerts '<Bar>'

someObject.myMethod.apply(someOtherObject, ['<', '>']); // alerts '<Bar>'

* **How to add a array element.**var tt=["ram","sumit"];  
  tt[2]="vinay";  
  console.log(tt);
* **What is the difference between window.onload and onDocumentReady in JavaScript?  
  window.onload**- Code in this method get executed when DOM tree is ready. But fact is code in this method executed when DOM tree is ready and All the external resource are load like Images, Flash vidoe or quicktime video. Loading of the external resources deplay execution of the actual script when page get displayed.  
  **$(document).ready**- Code in this method get executed once DOM tree loading is done. That means its not wait for the external resource get loaded. And run the javascript code which is in function.

**Describe the properties of an anonymous function in JavaScript?**An anonymous function is a function that was declared without any named identifier to refer to it. As such, an anonymous function is usually not accessible after its initial creation.  
**var** anon = **function**() {

alert('I am anonymous');

}

anon();

* **Use strict**JavaScript shows errors for a piece of codes, which did not show an error before. Strict mode also solves some mistakes that hamper the JavaScript engines to work efficiently.

**ARRAY OPERATION**

**\*. How to count element of array**

const data = [5, 5, 5, 2, 2, 2, 2, 2, 9, 4]  
function count(arr) {  
 return arr.reduce((prev, curr) => (prev[curr] = ++prev[curr] || 1, prev), {})  
}  
console.log(count(data))

* **how to append element in array at specific position**  
   **Append Single Element at a specific index** arrName.splice(1, 0,'newName1');  
   **Append Multiple Element at a specific index** arrName.splice(1, 0,'newElemenet1', 'newElemenet2', 'newElemenet3');  
  If we use offset value then it remove the value else the add only.   
  array.splice(start, deleteCount, item1, item2, item3 ...)

See all reference of array  
<https://www.tutorialspoint.com/javascript/javascript_arrays_object.htm>

* **How to convert array to string in javascript.** var arr = new Array("First","Second","Third");  
   var str = arr.join(" + ");  
   document.write("<br />str : " + str );   
   str : First + Second + Third
* **How to get the last index of given value in array**Javascript array lastIndexOf() method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at fromIndex.var index = [12, 5, 8, 130, 44].lastIndexOf(8); // 2  
  var index = [12, 5, 8, 130, 44, 5].lastIndexOf(5); // 5
* **How to reverse the array**

var arr = [0, 1, 2, 3].reverse();

3,2,1,0

* **How to sort array in javascript**Javascript array sort() method sorts the elements of an array.  
  var arr = new Array("orange", "mango", "banana", "sugar");   
   var sorted = arr.sort();  
  Returned array is : banana,mango,orange,sugar

**1) numeric sorting in ascending**  var points = [40, 100, 1, 5, 25, 10];  
 points.sort(function(a, b){return a-b});

**2) numeric sorting in descending**  var points = [40, 100, 1, 5, 25, 10];  
 points.sort(function(a, b){return b-a});

**3) alphabetically sorting in deccendig and descending** var fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.sort(); // accending  
 fruits.reverse(); // descending

\*\*\* understanding settimeout inside for loop

|  |
| --- |
| function doSetTimeout(i) {  setTimeout(function() { alert(i); }, 100);  }  for (var i = 1; i <= 2; ++i)  {  doSetTimeout(i);  } // 1 2  for (var i = 1; i <= 2; ++i)  {  setTimeout(function() { alert(i); }, 100);  } // 22  for (let i = 1; i <= 2; ++i)  {  setTimeout(function() { alert(i); }, 100);  } // 1 2  The short answer is that the for loop executes first, then it looks for the i value, which is 5, and then outputs four times, one for each loop iteration.  The setTimeout function can access the value of i because of closure. We ask for i within our console.log statement, but its value is set in an outer, enclosing scope, that of the for loop. Since inner functions have access to variables in an outer function, we are able to go up a level and retrieve the value of i, which is 5.  JavaScript is a single threaded single concurrent language which means it can only handle one task or piece of code at a time |

1. rule of functional component

2 way of updating store

3 getDeriveredstatefromeProps

4 context Api

5 var and let condole.log(a) console.log(b) var a=10 let b=20

6 for(var i=0 ; i <= i; i++) { setTimeOut(function(){ console.log(i) },0)

1

|  |
| --- |
| 1) function sum(){  alert(a);  var a=10;  }  //sum(); // undifined |

2

|  |
| --- |
| var a= 10 or a= 10  2) function sum(){  alert(a);  var a=10;  }  //sum(); // undifined |

var a= 10

2) function sum(){

var a=10;

alert(a);

}

//sum(); // undifined

3

|  |
| --- |
| **get sum(1,2,3,4,5...)** function sum(...arr){  return arr.reduce((pre,curr)=>(pre+curr));  }  **sum(1,2,3,4)** |

4

|  |
| --- |
| **Sum(1)(2)(3)()**  Let sum= function(a) { return function(b) {  If(b){  return sum(a+b);  } return a;  **add(1)(2,2));**  function add(...args) {  if (!args.length) return 0;  const result = args.reduce((accumulator, value) => accumulator + value, 0);  const sum = (...innerArgs) => {  if (innerArgs.length === 0) return result;  return add(...args, ...innerArgs);  };  return sum;  }  console.log(add(1)(2,2)); |

**Self invocated function ofr IIFE (immediate invocked function expresion)**

Self-invocation (also known as auto-invocation) is when a function executes immediately upon its definition. This is a core pattern and serves as the foundation for many other patterns of JavaScript development.

This is basically used for avoiding naming conflict as well as for achieving encapsulation. The variables or declared objects are not accessible outside this function.

IIFE (Immediately Invoked Function Expressions) is the best practice for writing scripts as plug-ins, add-ons, user scripts or whatever scripts are expected to work with other people's scripts. This ensures that any variable you define does not give undesired effects on other scripts.

to prevent pollution of the Global (or higher) scope.

It keeps code to a minimum

It’s also good for block scoping.

It’s good for encapsulation.

for( var i = 0; i < 10; i++ ) {

(function(num){

setTimeout(function(){

console.log(num)

})

})(i)

}

**Arrow function how to auto binding**

**function** Prefixer(prefix) {

**this**.prefix = prefix;

}

Prefixer.prototype.prefixArray = **function** (arr) { *// (A)*

'use strict';

**return** arr.map(**function** (x) { *// (B)*

*// Doesn’t work:*

**return** **this**.prefix + x; *// (C)*

});

};

> var pre = new Prefixer('Hi ');

> pre.prefixArray(['Joe', 'Alex'])

TypeError: Cannot read property 'prefix' of undefined

**13.2.1 Solution 1: that = this**

You can assign this to a variable that isn’t shadowed. That’s what’s done in line A, below:

**function** Prefixer(prefix) {

**this**.prefix = prefix;

}

Prefixer.prototype.prefixArray = **function** (arr) {

**var** that = **this**; *// (A)*

**return** arr.map(**function** (x) {

**return** that.prefix + x;

});

};

13.2.2 Solution 2: specifying a value for this

A few Array methods have an extra parameter for specifying the value that this should have when invoking the callback. That’s the last parameter in line A, below.

**function** Prefixer(prefix) {

**this**.prefix = prefix;

}

Prefixer.prototype.prefixArray = **function** (arr) {

**return** arr.map(**function** (x) {

**return** **this**.prefix + x;

}, **this**); *// (A)*

};

13.2.3 Solution 3: bind(this)

You can use the method bind() to convert a function whose this is determined by how it is called (via call(), a function call, a method call, etc.) to a function whose this is always the same fixed value. That’s what we are doing in line A, below.

**function** Prefixer(prefix) {

**this**.prefix = prefix;

}

Prefixer.prototype.prefixArray = **function** (arr) {

**return** arr.map(**function** (x) {

**return** **this**.prefix + x;

}.bind(**this**)); *// (A)*

};

#### 13.2.4 ECMAScript 6 solution: arrow functions [#](https://exploringjs.com/es6/ch_arrow-functions.html#_ecmascript-6-solution-arrow-functions)

Arrow functions work much like solution 3. However, it’s best to think of them as a new kind of functions that don’t lexically shadow this. That is, they are different from normal functions (you could even say that they do less). They are not normal functions plus binding.

With an arrow function, the code looks as follows.

**function** Prefixer(prefix) {

**this**.prefix = prefix;

}

Prefixer.prototype.prefixArray = **function** (arr) {

**return** arr.map((x) => {

**return** **this**.prefix + x;

});

};

Now, let’s see the difference between Synchronous and Asynchronous Transmission:

|  |  |  |
| --- | --- | --- |
| **S.NO** | **SYNCHRONOUS TRANSMISSION** | **ASYNCHRONOUS TRANSMISSION** |
| 1. | In Synchronous transmission, Data is sent in form of blocks or frames. | In asynchronous transmission, Data is sent in form of byte or character. |
| 2. | Synchronous transmission is fast. | Asynchronous transmission is slow. |
| 3. | Synchronous transmission is costly. | Asynchronous transmission economical. |
| 4. | In Synchronous transmission, time interval of transmission is constant. | In asynchronous transmission, time interval of transmission is not constant, it is random. |
| 5. | In Synchronous transmission, There is no gap present between data. | In asynchronous transmission, There is present gap between data. |

### **Redux-thunk**

Redux-thunk is the standard way of performing asynchronous operations in Redux. For our purposes, a thunk represents a function that is not called immediately, only when needed.

**Let x= 1+2**

The value 3 is assigned immediately to x.

**Let foo =() => 1+2;**

The sum operation is not executed immediately, only when you call foo(). This makes foo a thunk.

Redux-thunk allows an action creator to dispatch a function in addition to a plain object, converting the action creator into a thunk.

The advantage of using redux-thunk is that the component doesn’t know that it is executing an asynchronous action.

Since the middleware automatically passes the dispatch function to the function that the action creator returns, for the component, there will be no difference between asking to perform a synchronous and an asynchronous action (and they don’t have to care anyway). Since redux-thunk gives to the dispatched function the dispatch and getState methods from the store as parameters, you can also dispatch other actions and read the state to implement more complex business logic and workflows.