Interview Questions

What is prototype chain?

- Should have good understanding of prototype chain?

- whole concept of inheritance in JS is based on prototype
- Every object has Prototype, including Prototype object and that chain goes way back to the object that has no prototype.
- If you access any object value first it will look into its own proto if not found it will go back to proto chaining till it reach end or found that value

- Should know about "prototype" keyword

- Object does not have property prototype it has __proto__ which belongs to constructor or you can say its call
- You cannot set new
- Function have prototype
- Function prototype has constructor and __proto__

```
▪ constructor: ƒ one()
▪ __proto__:
    • constructor: ƒ Object()
    • hasOwnProperty: ƒ hasOwnProperty()
    • isPrototypeOf: ƒ isPrototypeOf()
    • propertyIsEnumerable: ƒ propertyIsEnumerable()
    • toLocaleString: ƒ toLocaleString()
    • toString: ƒ toString()
    • valueOf: ƒ valueOf()
    • __defineGetter__: ƒ __defineGetter__()
    • __defineSetter__: ƒ __defineSetter__()
    • __lookupGetter__: ƒ __lookupGetter__()
    • __lookupSetter__: ƒ __lookupSetter__()
    • get __proto__: ƒ __proto__()
    • set __proto__: ƒ __proto__()
    •
```

- Ex:

```
-
-    function one() {
-      this.tag = "one";
-    }
-    function two() {
-      one.call(this);
-      this.name = "two";
-    }
-
-    let t = new two();
```

-

- Should know about __proto__ keyword

- __proto__ has all the properties of its parent class
- For example, if you have created an array then __proto__ of that array will have all properties like length, index of.
- For object it has value tostring, hasownpropery, valueof, constructor

- If class A extends class B then what is the representation of instance of B (prototype chain)

```
-   class Person {
-       constructor(first, last, age, gender, interests) {
-         this.name = {
-           first,
-           last,
-         };
-         this.age = age;
-         this.gender = gender;
-         this.interests = interests;
-       }
-
-       greeting() {
-         console.log(`Hi! I'm ${this.name.first}`);
-       }
-
-       farewell() {
-         console.log(`${this.name.first} has left the building. Bye for now!`
    );
-       }
-     }
-     class Teacher extends Person {
-       constructor(first, last, age, gender, interests, subject, grade) {
-         super(first, last, age, gender, interests);
-         this.subject = subject;
-         this.grade = grade;
-       }
-     }
-     let snape = new Teacher(
-       "Severus",
-       "Snape",
-       58,
-       "male",
-       ["Potions"],
-       "Dark arts",
-       5
-     );
```

- In this case new object snape constructor is teacher and its __proto__ will be the person

What is curry function? Done

- Implementation of curry function
- Should run for single and multiple argument
- Solution: https://codesandbox.io/s/zen-tesla-dfdwv?file=/src/index.js

How do you assess performance of you code? Done

- Statements count as static constant
- Condition counts
- Loop counts (for, while, recursion)

How does connect function implementation look like in Redux?

- Create dummy connect function as HOC
- Should provide MapStateToPros and MapDispatch implementation
- Should provide store subscribe and unsubscribe

```
function connect(mapStateToProps, mapDispatchToProps) {
  return function (WrappedComponent) {
    return class extends React.Component {
      render() {
        return (
          <WrappedComponent
            {...this.props}
            {...mapStateToProps(store.getState(), this.props)}
            {...mapDispatchToProps(store.dispatch, this.props)}
          />
        )
      }

      componentDidMount() {
        this.unsubscribe = store.subscribe(this.handleChange.bind(this))
      }

      componentWillUnmount() {
        this.unsubscribe()
      }

      handleChange() {
        this.forceUpdate()
      }
    }
  }
}
```

-

-
- Should provide forceUpdate method call on state change.
- Solution: https://gist.github.com/gaearon/1d19088790e70ac32ea636c025ba424e

  High order function are those with operate on function either by taking in a parameter or return function.

  Connect: redux has a single store , and any react component can subscribe the redux store with the help of connect method or react-redux , component can subscribe slice part of store or while store and when its corresponding store part change component will get notified

Event Loop:

- Explain with the help of diagram
- Should know about set Timeout, promise roll in event-loop

  Promises have better priority than setTimeout callback function in event loop stack

- Should know about call stack
- Should not about Browser Object Model
- Prioritisation of next task in queue
- Watch Video: https://www.youtube.com/watch?v=8aGhZQkoFbQ
  - Web apis / cplus plus apis: ajax, settimeout, dom : these will run as separate task or thread -
  - Stack – every line of code
  - Event loop- job is to look at stack and task queue, if stack is empty it will move first element of queue into stack
  - Task queue- after web api complete its task it pushes cb on task queue

Stack and Queue

- LIFO vs FIFO

How to implement Array like Class that should work for following

```
/*
	let arr = new CustomArray();
	arr[5] = 10
	console.log(arr.length) // should be 6
*/
```

- Read about Symbole.iterator
- Read about Proxy (and performance issue with it)
- Solution: https://codesandbox.io/s/intelligent-wiles-2tcx5?file=/src/index.js

How to sort an Array in N level

- Recursion should be used
- Sort function of array should be use.

Implement inheritance using ES5.

- Can be created using Object.create()
- The **`Object.create()`** method creates a new object, using an existing object as the prototype of the newly created object.
- In this case proto will be the parent object.
- No function included
- Can be create using constructor
  - This is like creating two function parent and child, in child function call parent.apply so event child object will inherit parent class.
  - Done in function
- Should learn about performance of both.

Array Filter function poly-fill

```
Array.prototype.customFilter = function(predicate) {
    let output = []
    let len =  this.length;
    for(let i = 0; i < len; i++){
        if(predicate(this[i])){
            output.push(this[i]);
        }
    }
    return output;
}
let even = [1,2,3,4,8,7,9,10].customFilter(val => val % 2 === 0);
```

```
console.log(even);
```

Babble ES6 to ES5 of Following

```
//--- ES6 ------------input
let x = 10;
if(x == 10){
    let a = 20;
    console.log(a);
}
console.log(a);

//-------ES5-----------output
var x = 10;
if(x == 10){
    var _a = 20;
    console.log(_a);
}
console.log(a)
```

Result of filter function x => x

```
// input:
[1,3,2,0,-1,3].filter(x => x)

// output: [1,3,2,-1,3]
```

Following is the list of question that may be asked as well.

What is prototype chain?

- whole concept of inheritance in JS is based on prototype
- Every object has Prototype, including Prototype object and that chain goes way back to the object that has no prototype.

What is the difference between Call, Apply and Bind?

What is the difference between slice and splice?

`splice()` changes the original array whereas `slice()` doesn't but both of them returns array object

What is the difference between == and === operators?

== compares two variables irrespective of data type while **===** compares two variables in a strict check, which means it checks for data type also then it returns true or false.

- Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.
- Two numbers are strictly equal when they are numerically equal (have the same number value). `NaN` is not equal to anything, including `NaN`. Positive and negative zeros are equal to one another.
- Two Boolean operands are strictly equal if both are true or both are false.
- Two objects are strictly equal if they refer to the same `Object`.
- `Null` and `Undefined` types are `==` (but not `===`). [I.e. (`Null==Undefined`) is `true` but (`Null===Undefined`) is `false`]

What is lambda or arrow functions?

An **arrow function expression** is a syntactically compact alternative to a regular function expression, although without its own bindings to the `this`, `arguments`, `super`, or `new.target` keywords. Arrow function expressions are ill suited as methods, and they cannot be used as constructors.

What is a higher order function?

High order function is those with operate on function either by taking in a parameter or return function.

What is the difference between let and var?

Main difference is scoping rules. Variables declared by `var` keyword are scoped to the immediate function body (hence the function scope) while `let` variables are scoped to the immediate *enclosing* block denoted by `{ }` (hence the block scope).

Hoisting

While variables declared with `var` keyword are hoisted (initialized with `undefined` before the code is run) which means they are accessible in their enclosing scope even before they are declared:
`let` variables are not initialized until their definition is evaluated. Accessing them before the initialization results in a `ReferenceError`. Variable said to be in "temporal dead zone" from the start of the block until the initialization is processed.

At the top level, `let`, unlike `var`, does not create a property on the global object:

What are closures?

A **closure** is a feature in **JavaScript** where an inner function has access to the outer (enclosing) function's variables — a scope chain. ... it has access to its own scope — variables defined between its curly brackets. it has access to the outer function's variables. it has access to the global variables

What is a cookie?

1. Cookies are small strings of data that are stored directly in the browser. They are a part of HTTP protocol, defined by RFC 6265 specification.
2. Cookies are usually set by a webserver using response `Set-Cookie` HTTP-header. Then the browser automatically adds them to (almost) every request to the same domain using `Cookie` HTTP-header.
3. Properties are: expires, max-age, secure, http-only

What is a promise?

JavaScript | Promises. Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

What is typeof operator?

What is the difference between null and undefined?

In JavaScript, `undefined` means a variable has been declared but has not yet been assigned a value, such as:

What is the difference between window and document?

`Window` is the main JavaScript object root, aka the `global object` in a browser, also can be treated as the root of the document object model. You can access it as `window`
`window.screen` or just `screen` is a small information object about physical screen dimensions.
`window.document` or just `document` is the main object of the potentially visible (or better yet: rendered) document object model/DOM.
Since `window` is the global object you can reference any properties of it with just the property name - so you do not have to write down `window.` - it will be figured out by the runtime.

What is event bubbling?

1. When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.
2. Let's say we have 3 nested elements `FORM > DIV > P` with a handler on each of them.
3. A click on the inner `<p>` first runs `onclick`:
4. On that `<p>`.
5. Then on the outer `<div>`.
6. Then on the outer `<form>`.
7. And so on upwards till the `document` object
8. `event.target` – is the "target" element that initiated the event, it doesn't change through the bubbling process.
9. `this` – is the "current" element, the one that has a currently running handler on it.
10. `event.stopPropagation()`
11. `stopImmediatePropagation`

What is event capturing?

1. Capturing phase – the event goes down to the element.
2. That is: for a click on `<td>` the event first goes through the ancestors chain down to the element (capturing phase), then it reaches the target and triggers there (target phase), and then it goes up (bubbling phase), calling handlers on its way.
3. `elem.addEventListener(..., {capture: true})`

Is JavaScript a case-sensitive language?

1. **JavaScript** is a **case-sensitive language**. This means that **language** keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters. The while keyword, for example, must be typed "while", not "While" or "WHILE".

What is the use of preventDefault method?

1. The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.
2. For example, this can be useful when:

3. Clicking on a "Submit" button, prevent it from submitting a form
4. Clicking on a link, prevent the link from following the URL

What are break and continue statements?

1. Break exit the loop
2. Continue move code to next iterator

What is tree shaking?

Removal of dead code, can be done by webpack

What is a Regular Expression?

/pattern/modifiers;

What is a spread operator?

Spread syntax allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

What is for...of statement?

```
const array1 = ['a', 'b', 'c'];


for (const element of array1) {

  console.log(element);

}


// expected output: "a"

// expected output: "b"

// expected output: "c"
```

What is the difference between Shallow and Deep copy?

An object is said to be shallow copied when the source top-level properties are copied without any reference and there exists a source property whose value is an object and is copied as a reference.

Example: Using Object.assign() method

Deep copy of object

A deep copy will duplicate every object it encounters. The copy and the original object will not share anything, so it will be a copy of the original.

Example:Using JSON.parse(JSON.stringify(object));

What is a thunk function?

What is destructuring?

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

What is currying function?

What is a pure function?

The **function** always returns the same result if the same arguments are passed in. It does not depend on any state, or data, change during a program's execution.

What is IIFE(Immediately Invoked Function Expression)?

An **IIFE** (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.

```
(function () {
    statements
})();
```

What is Hoisting?

Hoisting is a **JavaScript** mechanism where variables and function declarations are moved to the top of their scope before code execution. Inevitably, this means that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.

What is scope in javascript?

Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to **JavaScript**. The two types of **scope** are local and global: Global variables are those declared outside of a block. Local variables are those declared inside of a block.

What is web storage?

With web storage, web applications can store data locally within the user's browser.

- `window.localStorage` - stores data with no expiration date
- `window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

What is a post message?

How do you delete a cookie?

What are the differences between cookie, local storage and session storage?

Differences between **cookies and localStorage**
Cookies are mainly for reading server-side, whereas **local storage** can only be read by the client-side . Apart from saving data, a big technical **difference** is the size of data you can store, and as I mentioned earlier **localStorage** gives you more to work with

Cookies are pretty much a convenient way to carry information from one session on a website to another, or between sessions on related websites, without having to burden a server machine with massive amounts of data storage

What are the three states of promise?

Resolve, reject, pending

What is promise.all

The **`Promise.all()`** method returns a single `Promise` that fulfills when all of the promises passed as an iterable have been fulfilled, the iterable contains no promises or the iterable contains promises that have been fulfilled and non-promises that have been returned. It rejects with the reason of the first promise that rejects or with the error caught by the first argument, if that argument has caught an error inside it using try/catch/throw blocks.

What is a strict mode in javascript?

How do you access history in javascript?

What is isNaN?

What are PWAs?

What is the purpose of clearTimeout method?

What is a debugger statement?

What is a freeze method?

What is an event loop?

What is call stack?

What is an event queue?

What are template literals?

What are dynamic imports?

What are the differences between promises and observables?

React Lifecycle methods/hooks ?

Mount phase

First three part of render phase will execute only one when component mount.

```
constructor
componentWillMount
componentDidMount
render
```

update phase:

componentwillreceiveprops

shouldcomponentupdate – you can add condition to avoid update by returning false

componentdwillupdate

componenetdidupdate

unmount phase:

componenetwillunmount

React hooks lifecycle :

Componenetdidmount – empty array

Componenetdidupdate – no arrayewr

Shouldcomponentupdate – use memo

```
1) deep comparison object  // write own function
2) flatten array // write own function
3) memorize a function // write own function

4)
let arr = [2,4,5,2,4,5,5,6,3];

let arr_set = [...new Set(arr)];

let foo = arr_set.reduce((a,b) => a+b, 0);

console.log(foo+arr_set.length); /// what will be output

5)react-redux connect
6)PubSub implementation
6)BEM methodology
```

Core JS:

- Polyfills of call, apply, bind, array.reduce, array.map, array.filter, etc(all array inbuilt functions)
- How inheritance happens in javascript
- What is V8 and how callstack and memory heaps works
- Explaining why JS is single-threaded and non-blocking
- What is closures, why is it important and real life example where closure is applied
- How JS engine garbage collects the memory
- Different types of module system JS works on?
- How variables/functions are hoisted? How It effects the memory heap? Why var, const and let important?
- Different types of design patterns in js and what are the use cases and when to implement which design patterns?
- What is execution context and how is it useful of memory efficiency?
- How ES6 syntax written in es5? Refer to Babel-Repl
- What are performance optimization methods? Memoization, curring, partial currying?
- Difference between functional programming and OOP? When and why is it necessary?
- What is the importance of “This” keyword?
- How async works in JS? Promises, asyn-await, traditional XMLHTTPRequest? Why are these used and when to use what?
- Cookies, local storage, session storage? What are these and why is it used?
- Difference between composition and inheritance?
- All the js loops(for, foreach, for-in, for-of, while)
- Java Scripts
- JS
- React
- Coding game round :
- Structured
- Problem Solving
- JS and React
- Advance problem solving
- Approach to the problem
- Basics of Java Script
- Deep Concepts of JS
- Oops -
- How JS works under the hood
- ES6 - how it works
- React- Redux is a plus
- Java Script expert
- How it works under the hood
- Clarity of thoughts
- Scores are shared with Interview Panel
- Practical Implementation

CURRY FUNCTION EXAMPLE :

```
function makeCurry(fun, ...rest) {
    return function (...innerRest) {
      if (fun.length === rest.length + innerRest.length) {
        return fun.apply(null, [...rest, ...innerRest]);
      } else {
        return makeCurry(fun, ...rest, ...innerRest);
      }
    };
  }

  let multiply = function (a, b, c) {
    return a * b * c;
  };

  let multiplyWithOneParam = function (a) {
    return a + 10;
  };

  var result = makeCurry(multiply)(10, 20)(5);
  console.log("Curry :: ", result);

  result = makeCurry(multiplyWithOneParam)(20);
  console.log("Curry :: ", result);
```

PUBSUB like redux

```
var pubSub = {
  subscribers: {},
  subscribe: function (event, callback) {
    let index;
    if (!subscribers[event]) {
      subscribers[event] = [];
    }
    index = subscribers[event].push(callback) - 1;

    return {
      unsubscribe() {
        subscribers[event].splice(index, 1);
      },
    };
  },
  publish: function (event, data) {
    if (!subscribers[event]) return;
    subscribers[event].forEach((subscriberCallback) =>
      subscriberCallback(data)
    );
  },
};

function publishEvent() {
  const data = {
    msg: "TOP SECRET DATA",
  };
  pubSub.publish("anEvent", data);
}
pubSub.subscribe("anEvent", (data) => {
  console.log(`"anEvent", was published with this data: "${data.msg}"`);
});
```

CORS :

Cross-origin resource sharing (**CORS**) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos.

"use strict" Directive

The `"use strict"` directive was new in ECMAScript version 5.

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

The purpose of `"use strict"` is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

All modern browsers support "use strict" except Internet Explorer 9 and lower:

Reference-counting garbage collection

his algorithm reduces the problem from determining whether or not an object is still needed to determining if an object still has any other objects referencing it. An object is said to be "garbage", or collectible if there are zero references pointing to it.

Limitation : circular references

Mark-and-sweep algorithm

This algorithm reduces the definition of "an object is no longer needed" to "an object is unreachable".

This algorithm assumes the knowledge of a set of objects called *roots.* In JavaScript, the root is the global object. Periodically, the garbage collector will start from these roots, find all objects that are referenced from these roots, then all objects referenced from these, etc. Starting from the roots, the garbage collector will thus find all *reachable* objects and collect all non-reachable objects.