**React Js**

**Q. What is React js**

Ans,

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

React allows developers to create large web applications that use data and can change over time without reloading the page. It aims primarily to provide speed, simplicity, and scalability. React processes only user interfaces in applications. This corresponds to View in the Model-View-Controller (MVC) pattern, and can be used in combination with other JavaScript libraries or frameworks in MVC,

**Q: What is vue.js.**

Ans: Vue.js is an open-source model–view–Model front end JavaScript framework for building user interfaces and single-page applications. It was created by Evan You and is maintained by him and the rest of the active core team members.

**Q: What is useState**

Ans: useState is a Hook that allows you to have state variables in functional components. You pass the initial state to this function and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.

**Q: What is virtual dom in react js**

Ans: Virtual DOM is a virtual representation of the real DOM. virtual DOM is a copy of real DOM. React uses virtual DOM to enhance its performance. It uses the observable to detect state and prop changes. React uses an efficient diff algorithm to compare the versions of virtual DOM

**Q: What is context API and how to write code.**

Ans: The Context API is a React structure that enables you to exchange unique details and assists in solving prop-drilling from all levels of your application.

import React, { createContext } from "react";

const UserContext = createContext({

logic

});

<UserContext.Provider value={your value}>

{children}

</UserContext.Provider>

**Q: What is Redux js and how to write code.**

Ans: Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces. Similar to Facebook's Flux architecture, it was created by Dan Abramov and Andrew Clark.

// 1. Create a basic Reducer

import React, { useReducer} from 'react';

const nameReducer = ( state, action ) => {

return currentState;

};

const [nameState, dispatch] = useReducer(nameReducer , currentState);

// 2. create a store

const store = redux.createStore( nameReducer );

**Q: Which is best between Context API and Redux**

Ans: The Context seems to be more comfortable and more flexible than Redux. You can choose and implement a custom wrapper for handling updates or call actions. Context with React Hooks is a robust feature that looks very well for maintenance and understanding data flow in the application.

**Redux** is used to manage the state of a React app in a centralized place. "State" simply refers to data you need to render the user interface correctly. Examples would be:

* Products in a shopping cart
* The information whether the user is waiting for a Http request to finish

**Context:** You can define the Context object in a separate file or right next to a component in a component file. You can also have multiple Context objects in one and the same app.

Context should be provided in a component that wraps all child components that eventually need access to the Context.

**Q: What is class and function based component?**

Ans: **Function :**  First of all, the clear difference is the syntax. Just like in their names, a functional component is just a plain JavaScript function that returns JSX. A class component is a JavaScript class that extends React.Component which has a render method.

import React from "react";

function FunctionalComponent() {

return <h1>Hello, world</h1>;

}

**Class**: The JSX to render will be returned inside the render method.On the other hand, when defining a class component, you have to make a class that extends React.Component. The JSX to render will be returned inside the render method.

import React, { Component } from "react";

class ClassComponent extends Component {

render() {

return <h1>Hello, world</h1>;

}

}

**Q: What is a component in React and how to write.**

Ans: Components are the building blocks of any React app and a typical React app will have many of these. Simply put, a component is a JavaScript class or function that optionally accepts inputs i.e. properties(props) and returns a React element that describes how a section of the UI (User Interface) should appear.

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

ReactDOM.render( <Welcome name="Bipin" />, document.getElementById('root')

);

**Q: What is useSelector and dispatch in React js.**

Ans: **useSelector :**  useSelector is a function that takes the current state as an argument and returns whatever data you want from it. It's very similar to mapStateToProps() and it allows you to store the return values inside a variable within the scope of your functional components instead of passing down as props.

import React from 'react'

import { useSelector } from 'react-redux'

const CounterComponent = () => {

const counter = useSelector((state) => state.counter)

return <div>{counter}</div>

}

**Dispatch**: dispatch is a function of the Redux store. You call the store. dispatch to dispatch an action. This is the only way to trigger a state change. With React Redux, your components never access the store directly - connect does it for you.

function Counter({ count, dispatch }) {

return (

<div>

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

<span>{count}</span>

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

</div>

)

}

**Q: How to create store in React js**

Ans: Creates a Redux store that holds the complete state tree of your app. There should only be a single store in your app.

import { createStore } from 'redux'

function todos(state = [], action) {

switch (action.type) {

case 'ADD\_TODO':

return state.concat([action.text])

default:

return state

}

}

const store = createStore(todos, ['Use Redux'])

store.dispatch({

type: 'ADD\_TODO',

text: 'Read the docs'

})

console.log(store.getState())

// [ 'Use Redux', 'Read the docs' ]

**Q: what is localStore and sectionStore**

Ans: **localStore :** The localStorage read-only property of the window interface allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions.

localStorage is similar to sessionStorage, except that while localStorage data has no expiration time, sessionStorage data gets cleared when the page session ends — that is, when the page is closed. (localStorage data for a document loaded in a "private browsing" or "incognito" session is cleared when the last "private" tab is closed.)

These being: setItem(), getItem(), removeItem(), and clear().

myStorage = window.localStorage;

localStorage.setItem('myCat', 'Tom');

const cat = localStorage.getItem('myCat');

localStorage.removeItem('myCat');

localStorage.clear();

**sectionStore:** Session storage is a popular choice when it comes to storing data on a browser. It enables developers to save and retrieve different values. Unlike local storage, session storage only keeps data for a particular session. The data is cleared once the user closes the browser window.

Session storage is a perfect alternative to cookies. Its syntax is quite straightforward. Beginners can easily learn and implement this storage. Session storage can also accommodate a huge amount of data. Most browsers, including Chrome and Firefox, can store about 10 MBs of data in session storage.

These being: setItem(), getItem(), removeItem(), and clear().

sessionStorage.setItem("name:, "John Smith");

var user = sessionStorage.getItem("name");

sessionStorage.remove("name");

sessionStorage.clear();

**Q: How to define/write localStore and sectionStore**

Ans:

These being: setItem(), getItem(), removeItem(), and clear().

myStorage = window.localStorage;

localStorage.setItem('myCat', 'Tom');

const cat = localStorage.getItem('myCat');

localStorage.removeItem('myCat');

localStorage.clear();

These being: setItem(), getItem(), removeItem(), and clear().

sessionStorage.setItem("name:, "John Smith");

var user = sessionStorage.getItem("name");

sessionStorage.remove("name");

sessionStorage.clear();

**Q. React hooks.**

Ans: Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes. (We don’t recommend rewriting your existing components overnight but you can start using Hooks in the new ones if you’d like.)

React provides a few built-in Hooks like useState. You can also create your own Hooks to reuse stateful behavior between different components. We’ll look at the built-in Hooks first.

* useState
* useEffect
* useReducer

**Q. How to get data to form parameters in function React js.**

Ans:

class NameForm extends React.Component {

handleSubmit = (event) => {

event.preventDefault()

console.log(event.target[0].value)

console.log(event.target.elements.username.value)

console.log(event.target.username.value)

console.log(this.inputNode.value)

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input

type="text"

name="username"

ref={node => (this.inputNode = node)}

/>

</label>

<button type="submit">Submit</button>

</form>

)

}

}

**Q. What are new things included and removed in React js?**

Ans:

**Q. How to create component in react js**

Ans:

const Bipin= (props) => {

const person = props.name;

return (

<div>

<h1>Hello {person}!!</h1>

</div>

)

}

**Q. What is the use of useState and what is this.**

Ans: useState is the React Hook that allows you to manage the state within functional components. useState, like all other hooks,

**Q. What is difference between state props in react js**

Ans:

**Material Design**

**Q: Material Design**

Ans,

Google created the material design as a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.

Material Design is a design language developed in 2014 by Google. Expanding upon the "card" motifs that debuted in Google Now, Material Design makes more liberal use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows.

1. Material design is a three-dimensional environment containing light, material, and cast shadows.

2. All material objects have x, y, and z dimensions.

3. All material objects have a single z-axis position.

**Angular js**

**Q. MVC pattern in angular js:**

MVC as it is popularly called is a software design pattern for developing web applications.

Ans: AngularJS is a JavaScript-based open-source front-end web application framework of dynamic web application development by google. This is single page application. But also you create multiple pages application.

**Javascript jQuery**

**Q. What are Local storage, Section storage and Cookies?**

Ans:

1. **Local storage:**The localStorage read-only property of the window interface allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions.

localStorage is similar to sessionStorage, except that while localStorage data has no expiration time, sessionStorage data gets cleared when the page session ends — that is, when the page is closed. (localStorage data for a document loaded in a "private browsing" or "incognito" session is cleared when the last "private" tab is closed.)

1. **Section storage sectionStore:**The read-only sessionStorage property accesses a session Storage object for the current origin. sessionStorage is similar to localStorage; the difference is that while data in localStorage doesn't expire, data in sessionStorage is cleared when the page session ends.
2. **Cookies:**HTTP cookies (also called web cookies, Internet cookies, browser cookies, or simply cookies) are small blocks of data created by a web server while a user is browsing a website and placed on the user's computer or other device by the user’s web browser. Cookies are placed on the device used to access a website, and more than one cookie may be placed on a user’s device during a session.

Cookies serve useful and sometimes essential functions on the web.

**Q. Closure and Relevant function.**

1. **Closure**: A closure is a combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, 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.

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

**Q. What is OOJS.**

Ans: OOjs (short for "Object-oriented JavaScript") is a JavaScript library for working with objects. Features include inheritance, mixins, static inheritance and additional utilities for working with objects and arrays. It also provides an EventEmitter mixin for event-driven programming, and a factory pattern for decoupling consumers of functionality from any particular class's implementation.

**Q. What is DOM travelling?**

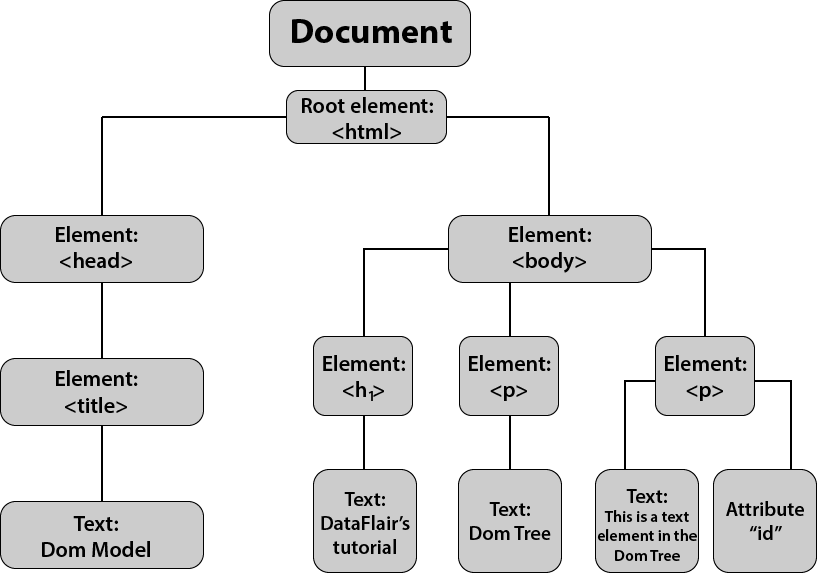
Ans: The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

The DOM allows us to do anything with elements and their contents, but first we need to reach the corresponding DOM object.

All operations on the DOM start with the document object. That’s the main “entry point” to DOM. From it we can access any node.

Here’s a picture of links that allow for travel between DOM nodes:



**Q. Jquery related all questions and function**

Ans:

**Q. How to reduce HTTP request**

Ans:

* Use fonts icons
* Use Minifying css and javascript
* Compress your images
* Remove extra tag elements or comments from the page.
* Delete unnecessary images
* Implement the lazy loading technique
* Use a Content Delivery Network (CDN)

**Q. Is it good to add jquery in the body tag?**

Ans: Yes

**Q. How to prevent extra features coming from another account (Skype, etc..)**

Ans:

<meta name="SKYPE\_TOOLBAR" content="SKYPE\_TOOLBAR\_PARSER\_COMPATIBLE" />

**Q. Disable and unable in the website (User, pages, guest, library etc..)**

Ans:

**Q. jQuery automation testing function and query**

Ans:

**Q. Automation testing function and query.**

Ans:

**Q. How to use automation testing**

Ans:

**Q. Have you written the automation function in code? for test code.**

Ans:

**Q. How to stop query/function etc in React js and jQuery.**

Ans: A simple way of how to stop a jQuery or JavaScript function in it’s tracks is to use the return false line. In most cases this is not the best way and there are other functions which should be used. Simple use the return false line to stop the function and return control to the calling function or page element.

**Q. How to travel data from child to parent and parent to children in Reactjs.**

Ans: **Data from child component to its parent:**  Following are the steps to pass data from child component to parent component:

* In the parent component, create a callback function. This callback function will retrieve the data from the child component.
* Pass the callback function to the child as a props from the parent component.
* The child component calls the parent callback function using props and passes the data to the parent component.

In Parent Component:

getData(val){

// do not forget to bind getData in constructor

console.log(val);

}

render(){

return(<Child sendData={this.getData}/>);

}

In Child Component:

demoMethod(){

this.props.sendData(value);

}

**Data from Parent component to its Child:** When you need to pass data from a parent to child class component, you do this by using props.

import React from 'react';

function Parent(){

const data = 'Data from parent';

return(

<div>

<Child dataParentToChild = {data}/>

</div>

)

}

function Child ({dataParentToChild}){

return(

<div>

{dataParentToChild}

</div>

)

}

export default Parent;

**Q. How to get data to form parameters in function jQuery.**

Ans: The serializeArray() method creates an array of objects (name and value) by serializing form values. This method can be used to get the form data.

$(selector).serializeArray()

<script>

$(document).ready(function() {

$("button").click(function() {

var x = $("form").serializeArray();

$.each(x, function(i, field) {

$("#output").append(field.name + ":"

+ field.value + " ");

});

});

});

</script>

**Q. What are new things included and removed in jQuery?**

Ans:

**New**:

* Arrow function
* Const
* Let
* modules,
* class declarations,
* lexical block scoping,
* iterators and generators,
* promises for asynchronous programming,
* destructuring patterns
* Classes
* Maps
* Sets
* Generators
* Proxies
* WeakMaps

**Remove:**

* **Var**

## **Q. What is Unit Testing?** [**Link**](https://www.xenonstack.com/insights/what-is-unit-testing)

**Ans:** Unit testing is a type of testing in which individual units or functions of [software testing](https://www.xenonstack.com/blog/software-testing-best-practices). Its primary purpose is to test each unit or function. A unit is the smallest testable part of an application. It mainly has one or a few inputs and produces a single output. In procedural programming, a unit referred to as an individual program, while object-oriented programming languages include Base/Superclass, abstract class, Derived/Child class takes place. Unit test frameworks, drivers, stubs and mocks /fake objects used in Unit Testing. It works on the basis of a White box technique.With Unit Testing Enteprises can -

1. Improve Quality of Code
2. Build Reusable and Reliable Code
3. Simplify Documentation
4. Enable Seamless Integration

**HTML and CSS**

**Q: amp:(inline CSS 120kb. Over right not CSS, add js slideshow with google, page size light weight. google search for the top.)**

Ans: The Accelerated Mobile Pages Project (AMP) is an open-source website publishing technology designed to improve the performance of web content and advertisements. The AMP Project led by Google is a competitor to Facebook's Instant Articles,[1] and includes several other large searches, social and web publishing platforms around the world.

**Q. HTML 5 API**

Ans: APIs (Application Programming Interfaces) are a way to create applications using pre-built components and are not unique to web development, or even to scripting languages.

Websites such as Twitter, YouTube and others provide APIs to the public so designers and developers can integrate features into their own websites (or for other purposes such as mobile or desktop applications).

**Q. What is Canvas**

Ans: The canvas element is part of HTML5 and allows for dynamic, scriptable rendering of 2D shapes and bitmap images. It is a low level, procedural model that updates a bitmap. HTML5 Canvas also helps in making 2D games.

**Q. What is CSS, and what is added and removed?**

Ans: CSS is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript

1. **New**: CSS3 is the latest version of the CSS language. This tutorial reviews main CSS3 features such as box shadows, fonts, rounded corners, selectors, and modules.
   1. Box Shadow
   2. Opacity
   3. Rounded Corners
   4. Attribute Selectors

**Q. What is accessibility?**

Ans: Accessibility is the practice of making your websites usable by as many people as possible. We traditionally think of this as being about people with disabilities, but the practice of making sites accessible also benefits other groups such as those using mobile devices, or those with slow network connections.

**Q. What is Semantic?**

Ans: Semantics is a linguistic concept separate from the concept of syntax, which is also often related to attributes of computer programming languages. The idea of semantics is that the linguistic representations or symbols support logical outcomes, as a set of words and phrases signify ideas to both humans and machines.

A semantic element clearly describes its meaning to both the browser and the developer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

* <article>
* <aside>
* <details>
* <figcaption>
* <figure>
* <footer>
* <header>
* <main>
* <mark>
* <nav>
* <section>
* <summary>
* <time>

**Q. Bootstrap related to all questions and tags.**

Ans: Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

**Q. What is SVG**

Ans: SVG stands for Scalable Vector Graphics SVG is used to define vector-based graphics for the Web SVG defines the graphics in XML format Every element and every attribute in SVG files can be animated

**Q. What are new things included and removed in HTML5, CSS3?**

Ans:

**HTML5** : Some elements are removed in HTML5, like isindex, noframes, acronym, applet, basefont, dir, font, frame, frameset, big, center, strike, and tt. HTML5 supports new kinds of form controls, including dates and times, email, number, range, tel, URL, search, etc. There are multiple new features and new elements in HTML5.

**CSS3** : CSS3 is the latest version of the CSS language. This tutorial reviews main CSS3 features such as box shadows, fonts, rounded corners, selectors, and modules.

Look at this list of CSS3 tutorials explaining the majority of modules:

* User Interface
* Box Model
* Selectors
* Backgrounds
* Borders
* Media Queries
* Transforms
* Text Effects
* Multiple Column Layout
* Animations

**Q. On which website you are finding the problem solution on the internet, google.**

Ans: Stackoverflow

**Q. Which website you are following for new things related to your fields.**

Ans:

1. css-tricks,
2. Javascript Info : <https://javascript.info/intro>
3. W3schools
4. Tutorials Points: <https://www.tutorialspoint.com/index.htm>

**Q.Regression Testing:**

Ans:

Regression testing is a software testing practice that ensures an application still functions as expected after any code changes, updates, or improvements.

All frameworks in CSS

1. how to design a website.
2. colour palette
3. Typographic
4. Responsive or not
5. gride
6. flow vertical alignment
7. age group
8. content information

**Q. What is ES6?**

Ans: ES6 is an acronym of ECMAScript 6 and also known as ECMAScript 2015.

ES6 or ECMAScript6 is a scripting language specification which is standardized by ECMAScript International. It is used by the applications to enable client-side scripting. This specification is affected by programming languages like Self, Perl, Python, Java, etc. This specification governs some languages such as JavaScript, ActionScript, and Jscript. ECMAScript is generally used for client-side scripting, and it is also used for writing server applications and services by using Node.js.

ES6 allows you to make the code more modern and readable. By using ES6 features, we write less and do more, so the term 'Write less, do more' suits ES6. ES6 introduces you many great features such as scope variable, arrow functions, template strings, class destructions, modules, etc.

ES6 was created to standardize JavaScript to help several independent implementations. Since the standard was first published, JavaScript has remained the well-known implementation of ECMAScript, comparison to other most famous implementations such as Jscript and ActionScript.

**Q. New feature of ES6 list**

Ans:

* The let keyword
* The const keyword
* Arrow Functions
* For/of
* Map Objects
* Set Objects
* Classes
* Promises
* Symbol
* Default Parameters
* Function Rest Parameter
* String.includes()
* String.startsWith()
* String.endsWith()
* Array.from()
* Array.keys()
* Array.find()
* Array.findIndex()
* New Math Methods
* New Number Properties
* New Number Methods
* New Global Methods
* Iterables Object.entries
* JavaScript Modules

**Q. What is the difference between var let and const in javascript.**

Ans:

var, let, and const are ways to declare variables.

It is important to use these declarations the right way because they were made to benefit the developer and make code easier to understand.

Scope is the way JavaScript finds declarations.

**There are Block scope and Function scope.**

**What is Block Scope:**

* A Block scope is defined with curly braces.
* Using var in a Block scope doesn’t prevent the variables from being accessible outside it.
* Using const or let in a Block scope prevents the variables from being accessible outside it.

{

var varDeclaration = "I am a var in a block scope";

console.log(varDeclaration); // I am a var in a block scope

}

// varDeclaration is accessible outside the block scope

console.log(varDeclaration); // I am a var in a block scope

{

let letDeclaration = "I am let in a block scoped";

console.log(letDeclaration); // I am let in a block scoped

}

console.log(letDeclaration);

// Uncaught ReferenceError: letDeclaration is not defined

{

const constDeclaration = "I am a const in a block scoped";

console.log(constDeclaration); // I am a const in a block scoped

}

console.log(constDeclaration);

// Uncaught ReferenceError: constDeclaration is not defined

Anything with a opening curly braces and closing curly brace creates a Block scopes.

Examples of Block scopes are if else statements and for loops.

**What is Function Scope:**

Function scope is anything inside a function. Everything declared inside a function isn’t accessible outside it.

const sayApples = "Apples from global scope";

function sayFruit () {

let sayApples = "Apples from sayFruit()";

const sayBananas = "Bananas from sayFruit()";

var sayOranges = "Oranges from sayFruit()";

console.log(sayApples);

}

sayFruit(); // Apples from sayFruit()

console.log(sayApples); // Apples from global scope

console.log(sayBananas);

// Uncaught ReferenceError: sayBananas is not defined

console.log(sayOranges);

// Uncaught ReferenceError: sayOranges is not defined

When executing sayFruit() “Apples from sayFruit()” is printed instead of “Apples from global scope” because the variable sayApples in the function is defined in the same scope as where console.log() is executed.

When looking for variables it starts at where the variable is referenced and it goes up scope until it reaches the global scope.

**Var**:

* var can be redeclared and reassigned.
* var is hoisted and initialized to undefined.
* var doesn’t care about block scope.
* If there is no var declaration JavaScript assumes you meant to make it a variable and puts it in the global scope.

**>> why var sucks:**

* it pollutes the global scope.
* Allows variables to be redeclared.
* The global scope version of the variable is changed.

**Let:**

let is usually the variable declaration that is used over var because it has better features than var does.

1. It can not be redeclared but can be redefined.

let favoriteJuice = "Orange juice";

favoriteJuice = "Apple juice";

console.log(favoriteJuice); // Apple juice

let favoriteJuice = "Cranberry juice";

// Uncaught SyntaxError: Identifier 'favoriteJuice' has already been declared

1. let is hoisted but not initialized.

console.log(hoisted);

// Uncaught ReferenceError: hoisted is not defined

let hoisted = "Hi I am hoisted but not initialized";

1. let uses block scope which means it can only be access in the scope it was defined in.

let favoriteFruit = "Oranges";

if (true) {

let favoriteFruit = "Bananas";

console.log(favoriteFruit); // Bananas

}

console.log(favoriteFruit); // Oranges

**>> why var sucks:**

1. let doesn’t pollute the global scope.

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

// code

}

console.log(i)

// Uncaught ReferenceError: i is not defined

1. let is Block scope which means the problems we had with var doesn’t apply to let.

**Const:**

variables declared with const should hold values that are constant.

1. const cannot be redeclared or reassigned.

const isJavaScriptCool = true;

isJavaScriptCool = false;

// Uncaught TypeError: Assignment to constant variable.

const isJavaScriptCool = "maybe";

// Uncaught SyntaxError: Identifier 'isJavaScriptCool' has already been declared

1. const must be initialized when declared.
2. Just like let const is hoisted but not initialized.

console.log(hoisted);

// Uncaught ReferenceError: hoisted is not defined

const hoisted = "Hi I am hoisted but not initialized";

1. const variable declarations are block scoped like let

if (true) {

const areYouAwesome = true;

console.log(areYouAwesome); // true

}

console.log(areYouAwesome);

// Uncaught ReferenceError: areYouAwesome is not defined

**>> const isn’t really constant**

* Some values can be changed with a const variable declaration.
* values can be changed in objects.
* Add, modify, and remove elements from an array.

**Most if not all the time use let and const.**

* let and var don’t have to be initialized when declared. const has to be initialized when declared.
* var can be redefined and redeclared, let can be redefined but not redeclared, const can’t be redefined or redeclared.
* var declarations are globally or function scoped while let and const are block scoped.
* use const when wanting to declare a variable that shouldn’t change.

**Q. What is Var, Var is function base or global base**

Ans: Function and global base, Define under function or outside function.

**Q. what is let, let is block scope and how, code**

Ans: Block Scope, Ex: loop and If else

**Q. What is const, Const is function base or global base**

Ans: Block Scope, Ex: loop and If else

**Q. What is block label scope/ conditional.**

Ans: This scope restricts the variable that is declared inside a specific block, from access by the outside of the block. The let & const keyword facilitates the variables to be block scoped. In order to access the variables of that specific block, we need to create an object for it. Variables declared with the var keyword, do not have block scope.

**Q. What is casting in javascript**

Ans: In programming language, casting is a way of telling the compiler to change an expression or value from one type to another. At times, you may want to convert your JavaScript expressions or values from one type to another.

**Q. javascript synchronous or asynchronous language**

Ans: JavaScript is synchronous. It is a blocking and single-threaded programming language. Means JavaScript can execute only one code block at a time.

**Q. What is synchronous in javascript**

Ans: Synchronous programming is the programming in which the program executes in the order it is written and only one code block is executed at a time.

In another word, synchronous programs are blocking programs which block execution of code blocks below it until the current code block finishes its execution. There is no parallel execution in synchronous programming, code executes one-by-one.

let today = new Date();

console.log(today);

for (let i = 0; i < 2; i++) {

console.log(i);

}

console.log("Hello World!");

**Q. What is asynchronous in javascript**

Ans:Asynchronous programming is the programming that executes in the order it is written until some asynchronous operation arrives.

When an asynchronous operation arrives then some block of code split from the program's normal execution and starts executing parallel to other blocks.

This means that instead of executing code blocks one by one it would be executing more than one block of code at a time.

let i = 0;

setTimeout(() => {

console.log("Time out!");

}, 2000);

setInterval(() => {

console.log(i++);

}, 1000);

console.log("Hello World!");

There are mainly 2 types of asynchronous code style in JavaScript:

* **Callback functions:**

Callback functions are the functions that are passed as an argument to another function.

* **JavaScript Promises**

**Q. How can we perform synchronous operation in javascript:**

Ans: At time we perform one action, We need to perform another action then we need to wait complete first.

**Q. Any asynchronous framework you know in javascript.**

Ans:

* Tornado
* Sanic
* Vibora
* Quart
* FastAPI

**Q. What is the use of flex-box.**

Ans: In a perfect world of browser support, the reason you'd choose to use flexbox is because you want to lay a collection of items out in one direction or another. As you lay out your items you want to control the dimensions of the items in that one dimension or control the spacing between items. These are the uses that flexbox was designed for.

**Q. Object-fit:**

Ans: The object-fit CSS property sets how the content of a replaced element, such as an <img> or <video> , should be resized to fit its container. You can alter the alignment of the replaced element's content object within the element's box using the object-position property.

**Q. Grid layout.**

Ans: The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning. [Exm](https://www.w3schools.com/css/css_grid.asp)

**Q. What is the difference between jQuery and React and Angular**

Ans:

JQuery is a Javascript library used to add more interactivity (animations, etc) to websites and manipulate DOM.

AngularJS is a Javascript MVC (Model-View-Controller) Framework used to make and design web apps easily.

ReactJS is a Javascript library that focus on building user interface of web apps. So, it only focus on the "V" of MVC.

**Q. What is closer in javascript**

Ans: **Closure**: A closure is a combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, 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.

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

**Q. What is the spread operator in javascript.**

Ans: The spread operator is a useful and quick syntax for adding items to arrays, combining arrays or objects, and spreading an array out into a function’s arguments.

JavaScript, spread syntax refers to the use of an three dots (…) to expand an iterable object into the list of arguments.

With Spread Operator, allow the expression to expand to multiple arguments, elements, variables, etc.

**Q. What is the arrow function?**

Ans: Arrow functions are anonymous functions (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined.

Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

**Q. What is difference between normal function and arrow function**

Ans: This is because a normal function’s this is bound to the object that calls the function. In contrast to a normal function, an arrow function’s this is always bound to the outer function that surrounds the inner function.

1. "this"

An arrow function does not have it's own this keyword. Instead, this is bound to the parent this. In the case where the arrow function is executed in the global context (it has no parent function/scope), this will be bound to the global object (window in the browser or globalThisin node).

2. Implicit return

One of (in my opinion) the best things about an arrow function is implicit returns. What is an implicit return? Take the example at the top of the post. If we have a function that is really only a one-liner, it'd be nice not to have to write out the curly brackets and returnstatement, right? Well, arrow functions allow u…

3. Always anonymous

The third major difference between arrow functions and regular functions is that arrow functions are always anonymous. They cannot be named. Whereas a regular function created with the function keyword can be named or anonymous (function myFunc() {... } vs function() { ... }),

Table of Contents

1. this value

Regular function

Arrow function

2. Constructors

Regular function

Arrow function

3. arguments object

Regular function

Arrow function

4. Implicit return

Regular function

Arrow function

5. Methods

Regular function

Arrow function

6. Summary