MERN STACK INTERVIEW QUESTIONS

# **HTML (Hypertext Markup Language)**

1. What is HTML and what is its role in web development?

HTML stands for Hypertext Markup Language. It is the standard markup language used for creating the structure and presentation of web pages. HTML provides a set of tags and elements that define the structure and content of a webpage. It uses a hierarchical structure to organize and display text, images, links, multimedia, and other elements on a web page.

1. Explain the basic structure of an HTML document.?

An HTML document follows a specific structure known as the basic HTML structure. It consists of the following components:

<!DOCTYPE html> <html> <head> <title>Title of the Document</title> </head> <body> Content of the document goes here. </body> </html>

Let's break down the components:

* The **<!DOCTYPE html>** declaration is the document type declaration and specifies the version of HTML being used. In this case, it indicates the use of HTML5.
* The **<html>** element serves as the root element and encompasses the entire HTML document.
* The **<head>** element contains meta-information about the document, such as the document's title, CSS stylesheets, JavaScript files, character encoding, and more. It is not displayed directly on the webpage.
* Within the **<head>** element, the **<title>** element specifies the title of the document, which is displayed on the browser's title bar or tab.
* The **<body>** element contains the main content of the HTML document, such as headings, paragraphs, images, links, and other elements. It is displayed directly on the webpage.

1. What is the purpose of the DOCTYPE declaration in HTML?

The DOCTYPE declaration, also known as the Document Type Declaration, is an important component of an HTML document. It is placed at the very beginning of the HTML document before the **<html>** tag. The DOCTYPE declaration informs the web browser or HTML parser about the version of HTML used in the document.

The main purpose of the DOCTYPE declaration is to ensure that the web browser renders the HTML document correctly, according to the specified version of HTML. It helps establish a standardized set of rules and syntax that the browser should follow when parsing and rendering the document.

1. What are the different versions of HTML you are familiar with?

* HTML4 (Hypertext Markup Language 4): HTML4 was a major version of HTML introduced in 1997. It introduced new features and improvements, such as the ability to include style sheets and scripts, support for tables, frames, and forms, and enhanced accessibility.
* HTML5 (Hypertext Markup Language 5): HTML5 is the latest version of HTML, introduced in 2014. It brought significant advancements to web development, including new semantic elements, native multimedia support, enhanced form controls, canvas for 2D drawing, improved APIs for offline web applications, and more.

1. Explain the difference between HTML and XHTML.

* HTML: HTML has a more lenient syntax, allowing for certain deviations and errors. For example, tags can be left unclosed, attribute values can be unquoted, and elements can be nested without strict rules.
* XHTML: XHTML follows the stricter syntax rules of XML. It requires well-formedness, meaning that all tags must be properly closed, attribute values must be enclosed in quotes, and nested elements must follow a specific order.

1. What is the purpose of the alt attribute in the **<img>** tag?

The **alt** attribute in the **<img>** tag is used to provide alternative text for an image. Here's the purpose and significance of the **alt** attribute:

1. How do you include an external CSS file in an HTML document?

<link rel="stylesheet" type="text/css" href="styles.css">

* The **rel** attribute specifies the relationship between the HTML document and the linked file. In this case, it should be set to "stylesheet" to indicate that the linked file is a CSS stylesheet.
* The **type** attribute defines the MIME type of the linked file, which should be set to "text/css" for CSS files.
* The **href** attribute specifies the path or URL of the external CSS file. Replace "styles.css" with the actual file path or URL relative to your HTML document.

1. Explain the concept of inline and block-level elements in HTML.
2. Inline Elements:
   * Inline elements are elements that do not create line breaks and typically flow within a line of text.
   * They are used to mark up smaller portions of text or elements within a larger block of content.
   * Examples of inline elements include **<span>**, **<a>**, **<strong>**, **<em>**, **<img>**, and **<input>**.
3. Block-Level Elements:
   * Block-level elements are elements that create line breaks before and after themselves, creating distinct blocks of content.
   * They are used to structure and define larger sections or divisions of a webpage.
   * Examples of block-level elements include **<div>**, **<p>**, **<h1>** to **<h6>**, **<ul>**, **<li>**, **<table>**, **<form>**, and **<section>**.
4. What are semantic elements in HTML? Provide some examples.

Semantic elements in HTML are tags that carry meaning and convey the purposeThese elements provide additional context and clarity to both humans and search engines about the organization and meaning of the content on a webpage. Here are some examples of semantic elements in HTML:

1. **<header>**: Represents the introductory content or a container for a group of introductory or navigational elements. It typically includes branding, logos, navigation menus, and headings.
2. **<nav>**: Defines a section of navigation links. It contains links that allow users to navigate within the website or to different sections of the page.
3. **<main>**: Represents the main content of a document or the dominant content within a webpage. It should be unique to the document and not duplicated.
4. **<article>**: Represents a self-contained composition within a document, such as a blog post, news article, or forum post. It can be independently distributed or syndicated.
5. **<section>**: Represents a standalone section or grouping of related content. It helps organize content into distinct sections or themes.
6. **<aside>**: Represents content that is tangentially related to the main content, such as sidebars, callouts, or advertisements.
7. **<footer>**: Represents the footer of a document or a section. It typically contains information about the author, copyright information, links to related documents, and contact details.
8. What is the difference between the **<div>** and **<span>** elements in HTML? When would you use each of them?

* The **<div>** element is a block-level element that is used to create a division or a container for grouping and structuring larger sections of content.
* It is typically used to create distinct sections or areas of a webpage and is commonly used for layout purposes.
* The **<span>** element is an inline-level element that is used to mark up or apply styling to smaller sections of content within a larger block of text or other elements.
* It is commonly used to apply CSS styles to specific parts of text, highlight or format inline content, or wrap elements like links or emphasis tags around specific words or phrases.

1. What are HTML entities? Give an example.

HTML entities are special characters or symbols that are represented using entity references in HTML

* **&nbsp;** for a non-breaking space.
* **&lt;** for the less-than symbol (<).
* **&gt;** for the greater-than symbol (>).
* **&amp;** for the ampersand symbol (&).
* **&quot;** for double quotation marks (").
* **&apos;** for apostrophe or single quotation mark (').

1. What is the purpose of the **<meta>** tag in HTML? Provide an example.

The **<meta>** tag in HTML is used to provide metadata or additional information about the HTML document. It includes various attributes that help browsers and search engines understand and process the webpage

1. What is the role of the **<head>** and **<body>** elements in an HTML document?

In summary, the **<head>** element contains metadata and other non-visible elements that provide information about the HTML document, while the **<body>** element contains the visible content and structure of the webpage that users see and interact with. Both elements play important roles in creating well-structured and informative HTML documents.

1. How do you add comments in HTML code?

<!-- This is a comment in HTML -->

1. Explain the purpose of the **<nav>** and **<article>** elements in HTML5.

The **<nav>** element represents a section of a webpage that contains navigation links or menus.

The **<article>** element represents a self-contained and independent piece of content within a webpage.

1. How do you create a responsive design in HTML using media queries?

Add the viewport meta tag: Place the following meta tag within the **<head>** section of your HTML document. This tag ensures that the browser renders the page properly on different devices.

@media (min-width: 601px) and (max-width: 1024px) {

  /\* Styles for medium screens \*/

}

# **CSS (Cascading Style Sheets)**

With CSS, you can define various styles for different elements of an HTML document, such as fonts, colors, layouts, spacing, borders, backgrounds, and more. It allows you to separate the content and structure of a web page from its presentation, providing better control over the visual aspects of the website.

1. What is CSS and what is its purpose?

* Styling
* Responsiveness
* Separation of the concerns

1. How do you include CSS styles in a web page?

There are three ways to do that

* Inline CSS
* Internal CSS
* External CSS

1. Explain the concept of CSS selectors and how they work.

Basic selector:

/\* Tag selector \*/

p {

  color: blue;

}

/\* Class selector \*/

.button {

  background-color: #FF5733;

}

/\* ID selector \*/

#header {

  font-size: 24px;

}

/\* Attribute selector \*/

input[type="text"] {

  border: 1px solid gray;

}

Combinators:

/\* Descendant combinator \*/

article p {

  font-style: italic;

}

/\* Child combinator \*/

ul > li {

  list-style-type: square;

}

/\* Adjacent sibling combinator \*/

h2 + p {

  font-weight: bold;

}

/\* General sibling combinator \*/

h3 ~ p {

  margin-left: 20px;

}

Pseudo-classes and Pseudo-elements:

/\* Pseudo-class \*/

a:hover {

  color: purple;

}

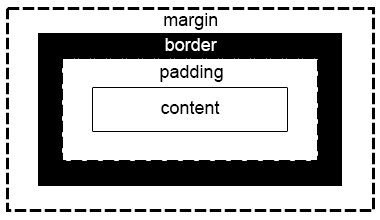
/\* Pseudo-element \*/

p::first-line {

  font-weight: bold;

}

1. What is the CSS box model? Describe its components.



1. How do you center an element horizontally and vertically in CSS?

Method 1: Using Flexbox

.container {

  display: flex;

  justify-content: center;

  align-items: center;

}

Method 2: Using Grid

.container {

  display: grid;

  place-items: center;

}

Method 3:

Using Absolute Positioning and Transforms

.container {

  position: relative;

}

.centered-element {

  position: absolute;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

}

1. Explain the importance of CSS specificity and how it affects style application.

CSS specificity is a calculation that determines which CSS rule takes precedence when multiple rules target the same element. Here's an explanation of the different levels of specificity and their impact, along with examples:

1. Inline Styles:

Inline styles have the highest specificity

<p style="color: red;">Inline Style</p>

1. ID Selectors

ID selectors have a higher specificity than class selectors or element selectors.

<p id="myId">ID Selector</p>

#myId {

color: blue;

}

1. Class Selectors, Attribute Selectors, and Pseudo-Classes:

<p class="myClass" data-custom="value">Class, Attribute, and Pseudo-Class Selectors</p>

.myClass {

  color: green;

}

[data-custom="value"] {

  color: purple;

}

p:first-child {

  color: orange;

}

1. Element Selectors and Pseudo-Elements:

Element selectors and pseudo-elements have the lowest specificity.

<p>Element Selector and Pseudo-Element</p>

p {

  color: gray;

}

p::first-line {

  text-transform: uppercase;

}

Summary

1. Inline CSS
2. ID Selector
3. Class Selector, attribute selector, and pseudo classes (like :hover)
4. Element Selector and pseudo Element (like ::before)
5. How do you handle cross-browser compatibility issues in CSS?\

* Basically, web kit question and by default browser has some margin and padding so to avoid that

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

.box {

  -webkit-transform: translateX(50%);

  -moz-transform: translateX(50%);

  -ms-transform: translateX(50%);

  transform: translateX(50%);

}

1. Explain the difference between 'inline', 'inline-block', 'block', and 'none' display values.

* Inline

Inline is a property or a value off display which means if element possessed inline value, then it possesses their own properties

It will Resist (can’t use)

🡺These following values will not apply

1. Width properties

* width
* min-width
* max-width

1. Height properties

* height
* min-height
* max-height

1. Margin properties

* margin-top
* margin-bottom

1. Padding properties

* padding-top
* padding-bottom
* Inline-block

“It will not disturb the old properties like the old value did not break the line so it will also do that but we can modify it and no resistance of values”

* Block

“Despite of inline and inline block it will break the line and shifted element to another and the remaining properties are same like inline block”

1. Explain the concept of CSS preprocessors (e.g., Sass, LESS) and their benefits.

Web kit usage decrees, use for big projects. There are the following functionalities preprocessor like sass provide

* Operators
* Variables
* Nesting
* Mixin
* Parameters
* Partials

1. How do you optimize CSS for better performance and faster loading times?

* Remove unnecessary code: white spaces, comments, and line breaks from your code because spaces, line breaks comments have its own memory so, to remove them give small size of file which increase downloading speed of code.
* Combine CSS files in one folder: This reduces the overhead of establishing separate connections for each file and speeds up loading times.
* Use CSS sprites: Place images as possible into single folder it is because when ever it needed it approach it.
* Avoid CSS imports: Minimize the use of CSS import statements, if computer

1. What are CSS vendor prefixes? Why are they used? Provide examples.

CSS vendor prefixes are webkits

1. How can you hide an element visually without affecting its layout in CSS?

Display: none; is used for visually hide the element without affecting its layout

Visibility: hidden; is use for visually hide the element but it will affect the layout

1. Explain the 'z-index' property and its role in CSS stacking context.

The **z-index** value: Elements with a higher **z-index** value will be placed in front of elements with a lower **z-index** value within the same stacking context.

1. How do you create a dropdown menu using CSS?

Ul element consist multiple children of li and have the display none value, after hover it will be display block

1. What are the different ways to apply CSS styles conditionally based on device or screen size?

* Media Queries
* CSS Grid and Flexbox
* Viewports Unites
* JavaScript and CSS
* CSS Frameworks

1. Explain the difference between 'em' and 'rem' units in CSS and when to use each.

* It is used to adjust child according to parents
* The **rem** unit represents the font size of the root element (**html**), and you can set it by as default value if considered 100% of an element

16px = 100%

1px = 100/16%

1px = 6.25%

10px = 62.5%

  html{

    font-size: 62.5%;

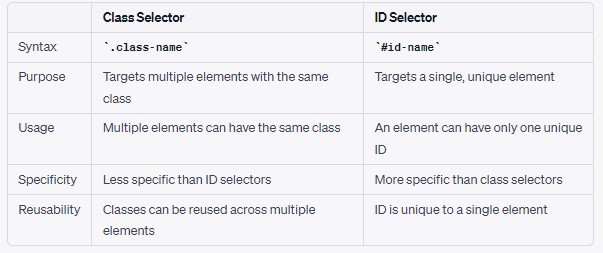
  }

It means you set root to 62.5% which mean default values set 10px instead of 16px

Now if you give 1em to any element then it will be considered as 10px because it drives from parent and in this case, parent is html if there was another element and you set its root something then child element will drive value because of em

In short once you set the root (rem) value of parent div then its child div driver the save value because of (em)

1. Explain the difference between a class selector and an ID selector in CSS.



1. Describe the concept of CSS sprites and their advantages.

<https://www.youtube.com/watch?v=A-AL5lhK2pY&ab_channel=TechGun>

1. How do you create a sticky/fixed navigation bar using CSS?

Show them some examples it is easy

1. What is the 'box-shadow' property in CSS? How can you use it to create a shadow effect?

box-shadow: h-offset v-offset blur spread color inset;

1. Explain the difference between the ':nth-child' and ':nth-of-type' pseudo-classes in CSS.

Syntax ka he farak hai is main

1. Describe the purpose of the 'content' property in CSS and how it is used.

It is used as a conjunction If you write something let’s suppose Adil then using ::before using content: ”Muhammad”; you can add Muhammad and it will become Muhammad Adil and with ::after using content: ”Younas” ; you will be able to add after text something now, it has become Muhammad Adil Younas

1. How can you make an element visually disappear from the page using CSS?

I will use display none and visibility hidden property

# **JavaScript**

1. What is JavaScript? Explain its basic features and capabilities.

A scripting language is a programming language that is designed to be interpreted and executed in real-time, typically without the need for compiling.

* Interpreted Execution which means JavaScript read line by line if it sees error code will not execute after that
* Dynamic typing eat you don’t have to worry about the type of data but type script has its own boundaries like variable declaration
* Rapid development due to easy syntax
* JavaScript have many libraries and frameworks which made it work easy

1. Differentiate between JavaScript and Java.

* JavaScript run on browser and java designed to run platform independent
* Both have OOP (object-oriented programming)
* Declaration of variables in JavaScript is easy but in java, use have to declare the type of the variable
* Both can be used on front end and back-end development

1. How do you include external JavaScript files in an HTML document?

* To include external JavaScript files in an HTML document, you can use the **<script>** tag with the **src** attribute

1. Describe the different data types in JavaScript.

There are two types of data in JavaScript

1. Primitive data types

In primitive data type when you assign a value to another variable then it would be equal to its value e.g, which is OK

let a = 45

let b = a  //b = 45

b = 100

console.log(a);  //45

console.log(b);   //100

* String (data wrap in single or double quotes)
* Number
* Boolean (true, false)
* Undefined (missing value)
* Null (nothing inside)

1. Non primitive data types

Non primitive data types have a small problem, It will store as a pointing to original

let a = [45,"adilyounas"]

let b = a

b[0] = 60

console.log(a);  //[60, 'adilyounas']

console.log(b);   //[60, 'adilyounas']

As you see in the above result if b change then a change which is not right so the solution is de-structuring

let a = [45,"adilyounas"]

let b = [...a]

b[0] = 60

console.log(a);  //[45, 'adilyounas']

console.log(b);   //[60, 'adilyounas']

* Object (data store in it as key value pair)
* Array (data store in it as index)
* Date

1. What is the control flow statements in JavaScript? Provide examples of each.

In JavaScript, control flow statements are used to control the execution flow of a program. They determine which code blocks are executed and in what order based on specified

* If……else
* Switch
* For
* While
* Do…. while
* Try…catch

1. How do you handle errors and exceptions in JavaScript?

In programming, an exception refers to an event or condition that occurs during the execution of a program

try {

  // Code that may throw an exception

} catch (error) {

  // Handle the exception

} finally {

  // Cleanup tasks

}

1. What are the differences between 'null' and 'undefined' in JavaScript?

* Null means empty and undefined means missing

1. How do you create and use functions in JavaScript? Explain the concepts of function declaration and function expression.

In JavaScript, functions are blocks of reusable code that can be defined and called to perform a specific task.

//Declare and use function

//if you declare function or variable using var then it contain globle scope otherwise script scope

// 1. Function declaration Function declarations are hoisted in JavaScript

function test1 (){

    console.log("Test1");

}

// 2. Function Expression

var test2 = function(){

    console.log("test2");

}

//3. Fat Arrow function

var test3 = ()=>{

    console.log("test3");

}

test1()  //test1

test2()  //test2

test3()  //test3

1. What is hoisting in JavaScript? Explain how it affects variable and function declarations.

Hoisting

“Hoisting is a behavior in JavaScript where variable and function declarations are moved to the top of their containing scope during the compilation phase before the code is executed”.

// HOISTING

//We will discuss about the globle execution context, call stack, funtion pop in, function pop out, let and const are not part of globle execution context they will undefined in script and js set values before code run which is undefined to variable and for pure function it considered it as it

//Example one

let a = 1

const b = 2

var c = 3

console.log(a); //1

console.log(b); //2

console.log(c); //3

// Example 2

console.log(a);  //Uncaught ReferenceError: Cannot access 'a' before initialization

console.log(b);  //Uncaught ReferenceError: Cannot access 'b' before initialization

console.log(c);  //undefined

let a = 1

const b = 2

var c = 3

//Declare and use function

//if you declare function or variable using var then it contain globle scope otherwise script scope

// 1. Function declaration Function declarations are hoisted in JavaScript

function test1 (){

    console.log("Test1");

}

// 2. Function Expression

var test2 = function(){

    console.log("test2");

}

//3. Fat Arrow function

var test3 = ()=>{

    console.log("test3");

}

//4 annonymous function

Function(){

}

test1()  //test1

test2()  //test2

test3()  //test3

1. Explain the concept of closures in JavaScript and provide an example.

//# Closure (function values ly kr aye ga apny lexical scope se)

// the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment).

var sum = function(a){

    console.log("Live Viewers " + a);

    var c =4

    return function(b){

        return a+b+c

    }

}

//a,c ki values function ly kr aya hai apny lexical scope se

//dosri languages k andar asa nahi hotaa

var store = sum(200)

console.log(store(10));

// Example2

var sum = function(a,b,c){

    return{

        getSumTwo:function(){

            return a+ b

        },

        getSumThree:function(){

            return a+b+c

        }

    }

}

// a,b,c ki values function k sath store2 k pass i hn

var store2 = sum(1,2,3)

console.log(store2.getSumTwo());

console.log(store2.getSumThree());

1. Explain the concept of lexical scope in JavaScript is lexical scope and closure are same or different.?

🡺“Closure is a function that retains access to variables from its lexical scope, even after the execution of the function has finished.”

🡺“Lexical Scope: The scope of variables determined by their position in the nested structure of blocks or functions in the source code. Child scopes have access to variables in their parent scopes, but not the other way around.”

1. Explain the concept of function currying in JavaScript

“When we receive function over function then it became hard to give them argument and receive parameters so function currying used there”

// if you get function over function then it is hard like this

function getName(a) {

  return function (b) {

    return function (c) {

      return a + b + c;

    };

  };

}

const d1 = getName(1);

const d2 = d1(1);

const d3 = d2(1);

console.log(d3); //3

//same example but this type we use function currying

function getName(a) {

  return function (b) {

    return function (c) {

      return a + b + c;

    };

  };

}

const data1 = getName(1)(1)(1);

console.log(data1); //3

1. What is the purpose of the 'this' keyword in JavaScript? How is it determined in different contexts?

* Alone, function, method, on function using strict mode

// # this keyword in JavaScript and globle object in js

// global object is the 'window' object in browsers and 'global' object in Node.js.

// function will give us the name Adil becaue name is present in his lexical environment or its lexical environment is globle in htis case

var name = "Adil"

function getName(){

    console.log(name);

}

getName()

// same example with this keyword

let father = "younas"

function getFather (){

    console.log(father);

}

this.getFather()

//SAME EXAMPLE WITH THIS KEYWORD AND ARGUMENT

const mother = "shafqat"

function getMother(mother){

    console.log(mother);

}

this.getMother("Shafqat")

//!THE ABOVE EXAMPLES ARE FOR GLOBLE OBJECT NOW WE ARE TALKING ABOUT THIS IN DETAILS

// 1. use "this" alone refer to window object in browser remember globle word is use is node and window in browser

console.log(this); //window object

// 2. Use in function also refer to "window object"

function testingThis(){

console.log(this);

}

testingThis() //window object

//3. as a method in a object it refers to the object

var obj = {

    name:"Adil younas",

    father:"Younas",

    mother:"Shafqat Azeem",

    showNames:function(){

        console.log(this);

    }

}

obj.showNames()

//return {name:"Adil younas",father:"Younas",mother:"Shafqat Azeem"}

//it means OOP is methods in a object

// 4. in strict mood only in function this is "undefined"

"use strict"

function useStrict (){

    console.log(this);

}

useStrict()  //undefined

1. What are JavaScript prototypes and how do they work?

“In JavaScript, prototypes are a fundamental mechanism that enables object-oriented programming and property inheritance. Prototypes allow objects to inherit properties and methods from other objects, creating a prototype chain that forms the basis of JavaScript's object-oriented nature”.

* **Prototype Object:**

Every object in JavaScript has an internal property called [[Prototype]] (also known as the "dunder prototype" or "internal prototype link"). This property references another object, which is the object's prototype. When you access a property or method on an object, and it doesn't exist on that object, JavaScript automatically looks up the prototype chain until it finds the property or reaches the end of the chain.

* **Prototype Chain:**

The prototype chain is a series of links from one object's prototype to the next. It forms the inheritance hierarchy, allowing objects to inherit properties and methods from their prototypes. When you access a property or method on an object, JavaScript first checks if the property exists on the object itself. If not, it looks up the prototype chain to find the property on the object's prototype. If it still doesn't find the property, it continues traversing the chain until it finds the property or reaches the end of the chain (the prototype with a null value).

array.\_\_proto\_\_

// [constructor: ƒ, at: ƒ, concat: ƒ, copyWithin: ƒ, fill: ƒ, …]

array.\_\_proto\_\_.\_\_proto\_\_

// {constructor: ƒ, \_\_defineGetter\_\_: ƒ, \_\_defineSetter\_\_: ƒ, hasOwnProperty: ƒ, \_\_lookupGetter\_\_: ƒ, …}

array.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_

null

* **Before you start prototype you need to understand**

Arrays are a specialized type of object in JavaScript. They are objects that have numeric keys (indexes) and a length property. Arrays can also have additional methods like push(), pop(), forEach(), etc., which are not available to regular objects.

const array = new Array()

console.log(typeof array); //Type of array is object it means prototype is a object attached with object. In simple word why object will add to the Array. The object only attached with object that is why people say everything in JS is object

e.g.

function greet() {

    console.log("Hello!");

  }

  greet.message = "Welcome!";

  console.log(greet.message); // Output: "Welcome!"

* **Myths About Array and object**

Note: As in Js typeof Array and object are object it means you can add methods in both

//Inside Array or object if there is a function then it is called its method. Array and Object have their own properties and methods separately

const array = [function(){console.log("Inside Array");}  ]

array[0]()  //Inside Array

array.poop = function(){

    console.log("Adil youans");

}

array.poop() //Adil younas

const obj = {fun:function(){console.log("Object inside method");}}

obj.fun()

obj.add = function(){

    console.log("Add in object");

}

obj.add() //Add in object

Now Prototype Inheritance

Note: JavaScript automatically looks up the prototype chain until it finds the property or reaches the end of the chain.

const obj = {

    name:"adil younas",

}

console.log(obj);

const obj2 = {

    roleNumber:1,

    \_\_proto\_\_:obj,

}

console.log(obj2.name); //adil younas

// What if obj2 have also name property

const obj = {

    name:"adil younas",

}

console.log(obj);

const obj2 = {

    roleNumber:1,

    name:"abeeha younas",

    \_\_proto\_\_:obj,

}

console.log(obj2.name); //abeeha younas

//What if obj2 call obj method but

const obj = {

    name:"adil younas",

    fun:function(){

        return this.name

    }

}

console.log(obj);

const obj2 = {

    roleNumber:1,

    name:"abeeha younas",

    \_\_proto\_\_:obj,

}

//pehly apna name dekha phir protoype ki Hierarchy wise up jaon ga

console.log(obj2.fun()); //abeeha younas

Pollyfill (Editing in Orignal Prototype)

//# Polyfill

// \_\_proto\_\_ is for inheritance and prototype is for adding new properties in Orignal Array or Object

Array.prototype.show= function(){

    return this

}

const mutantArray = ["Lahore","Faislabad"]

console.log(mutantArray.show());  //["Lahore","Faislabad"]

//What if i want to create my array to object converter function in Array prototype

const mutantArray = ["Lahore","Faislabad"]

Array.prototype.convertIntoObject = function(){

    let obj = {}

    this.forEach((element)=>{

       obj[element] = element

    })

    return obj

}

console.log(mutantArray.convertIntoObject());

1. Write a code to convert Array into object and object to Array

Array 🡺 Object

function arrayToObject(arr) {

  let obj = {};

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

    const key = arr[i];

    const value = arr[i + 1];

    obj[key] = value;

  }

  return obj;

}

const arr = ["name", "adil", "age", 45, "location", "New york"];

const result = arrayToObject(arr);

console.log(result);  //{name: 'adil', age: 45, location: 'New york'}

Object 🡺 Array

function objectToArray(obj) {

  const resultArray = [];

  for (const key in obj) {

    if (obj.hasOwnProperty(key)) {

      resultArray.push(key, obj[key]);

    }

  }

  return resultArray;

}

// Example usage:

const obj = { name: "John", age: 30, location: "New York" };

const arr = objectToArray(obj);

console.log(arr); // ['name', 'John', 'age', 30, 'location', 'New York']

1. Write code to sort string as well as numbers

//1. Sorting string of array in A-Z Manner

const fruits = ["banana", "apple", "orange", "grapes", "watermelon"];

fruits.sort();

console.log(fruits); // ['apple', 'banana', 'grapes', 'orange', 'watermelon']

//2. Sorting string of array in Z-A Manner

const fruits2 = ["banana", "apple", "orange", "grapes", "watermelon"];

fruits2.sort().reverse();

console.log(fruits2); //  ['watermelon', 'orange', 'grapes', 'banana', 'apple']

//3. Sorting number from 1-100 Manner

const numbers = [9, 3, 4, 5, 7, 6, 3, 1, 2];

numbers.sort((a, b) => a - b);

console.log(numbers);

//4. Sorting number from 100-1 Manner

const numbers2 = [9, 3, 4, 5, 7, 6, 3, 1, 2];

numbers.sort((a, b) => b - a);

console.log(numbers);

1. How do you manipulate the DOM (Document Object Model) using JavaScript?

Manipulating the DOM (Document Object Model) using JavaScript allows you to interact with and modify the content and structure of an HTML document dynamically. Here are common ways to manipulate the DOM using JavaScript

* **Accessing Elements:**

You can use various methods to select elements from the DOM, such as document.getElementById(), document.getElementsByClassName(), document.getElementsByTagName(), and document.querySelector(). These methods allow you to obtain references to elements so you can work with them.

<div id="myDiv">Hello, world!</div>

var divElement = document.getElementById("myDiv");

console.log(divElement.textContent); // Output: "Hello, world!"

* **Changing Content:**

You can modify the content of elements using properties like textContent, innerHTML, and innerText

<div id="myDiv">Hello, world!</div>

var divElement = document.getElementById("myDiv");

divElement.textContent = "Welcome!";

* **Modifying Styles:**

<div id="myDiv">Hello, world!</div>

var divElement = document.getElementById("myDiv");

divElement.style.color = "red";

divElement.style.backgroundColor = "yellow";

* **Adding/Removing Classes:**

You can add or remove CSS classes to/from elements using classList methods like add(), remove(), toggle(), and contains().

<div id="myDiv" class="box">Hello, world!</div>

var divElement = document.getElementById("myDiv");

divElement.classList.add("highlight"); // Adds the "highlight" class

divElement.classList.remove("box"); // Removes the "box" class

* **Creating New Elements:**

<div id="parentDiv"></div>

var parentDiv = document.getElementById("parentDiv");

var newParagraph = document.createElement("p");

newParagraph.textContent = "This is a new paragraph.";

parentDiv.appendChild(newParagraph);

* **Event Handling:**

You can attach event listeners to elements to respond to user interactions (e.g., clicks, keypresses). Use methods like addEventListener() to bind event handlers to elements.

<button id="myButton">Click Me</button>

var button = document.getElementById("myButton");

button.addEventListener("click", function() {

  console.log("Button clicked!");

});

1. Explain the concept of event handling in JavaScript and provide an example.

* **For Button**

<button id="myButton">Click Me</button>

var button = document.getElementById("myButton");

button.addEventListener("click", function() {

  console.log("Button clicked!");

});

* **For Form**

<form id="myForm">

  <input type="text" name="username" />

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

</form>

var form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {

  event.preventDefault(); // Prevent form submission

  var username = event.target.elements.username.value;

  console.log("Submitted by: " + username);

});

1. What are JavaScript promises? How do they help with asynchronous programming?

const fetchData = async () => {

  try {

    const response = await fetch("https://jsonplaceholder.typicode.com/todos");

    const data = await response.json();

    console.log(data);

  } catch (error) {

    console.error(error);

  }

};

fetchData();

1. Explain the concept of callbacks and how they are used in JavaScript.

<button id="myButton">Click Me</button>

var button = document.getElementById("myButton");

button.addEventListener("click", function() {

  console.log("Button clicked!");

});

1. What are arrow functions in JavaScript? How do they differ from regular functions?

//1. Syntex wise awsome

//2. By using fat arrow function you can't use this keyword in object method

//3. Don't need to return write code in front of =>

const person = {

  name: "John",

  greet: function () {

    // Regular function: 'this' refers to the object 'person'

    console.log("regular" + this.name); // Output: "regular", "John"

  },

  greetArrow: () => {

    console.log("fat" + this.name); // Output: "fat",""

  },

};

person.greet();

person.greetArrow();

1. Describe the differences between 'let', 'const', and 'var' for variable declaration in JavaScript.

In JavaScript, there are three keywords used for variable declaration: let, const, and var.

* **'var' (Function Scope): only function can stop him**
* Variables declared with var have function scope, meaning they are accessible within the function where they are declared or globally if declared outside any function.
* If a var variable is declared inside a block (e.g., an if statement or loop), it will still be accessible outside that block.
* var variables are hoisted, meaning they are moved to the top of their scope during the execution phase, allowing you to access them before their declaration.

function exampleFunction() {

  var x = 10;

  if (true) {

    var y = 20;

    console.log(x); // Output: 10

  }

  console.log(y); // Output: 20

}

* **'let' (Block Scope): block { } can stop him**
* Variables declared with let have block scope, meaning they are accessible only within the block (e.g., inside an if statement or loop) where they are declared.
* let variables are not hoisted, so they cannot be accessed before their declaration.
* If a let variable is declared within a block, it will not be accessible outside that block.

{

  let a = "Adil younas";

  {

    let b = 45

    console.log(a); //Adil younas

  }

  console.log(b); //Uncaught ReferenceError: b is not defined

}

* **'const' (Block Scope, Immutable):**
* Variables declared with const are similar to let in terms of block scope, but they have an additional characteristic of being immutable, meaning their value cannot be reassigned after initialization.
* When declaring a variable with const, you must assign a value to it at the same time. Once assigned, you cannot change its value.

// 1. Block scope

{

  const a = "Adil younas";

  {

    const b = 45

    console.log(a); //Adil younas

  }

  console.log(b); //Uncaught ReferenceError: b is not defined

}

//2. Assigning after initializing is not acceptable

const x = 10 // good

const y

y=20       //not good

//3. unable to reasign value

const obj = {

    name:"adil younas",

    father:"Younas"

}

obj.name = "Abeeha younas"

console.log(obj.name); //Abeeha younas

obj = {name:"Aqil younas",father:"Younas"}

//Uncaught TypeError: Assignment to constant variable.

1. Explain the concept of modules in JavaScript and how they help with code organization and reusability.

“In JavaScript, modules are a way to organize code into smaller, independent, and reusable pieces. They help in breaking down a large codebase into smaller, manageable components.”

export function add(a, b) {

  return a + b;

}

import { add } from "./math.js";

1. Explain the concept of event bubbling and event capturing in JavaScript also describe propagation and how to stop propagation

* Propagation is like a term you can use for traveling from child to parent or parent to child in the DOM Hierarchy
* When events travel upward from child to parent it is called event bubbling
* When events travel downward from parent to child then it is called event capturing
* If third parameter of addEventListener is false (by default value) then it indicates event bubbling and if true then it indicate event capturing
* We want to stop propagation then we have to use e.stopPropagation() inside the addEventListener callback function

  <div class="grandParent">

      <div class="parent">

        <div class="child"></div>

      </div>

    </div>

<style>

    \* {

      margin: 0;

      padding: 0;

      box-sizing: border-box;

    }

    body {

      display: flex;

      justify-content: center;

      align-items: center;

      width: 100%;

      height: 100vh;

    }

    .grandParent {

      width: 80vw;

      height: 80vh;

      border: 5px solid red;

      display: flex;

      justify-content: center;

      align-items: center;

    }

    .parent {

      width: 60vw;

      height: 60vh;

      border: 5px solid red;

      display: flex;

      justify-content: center;

      align-items: center;

    }

    .child {

      width: 40vw;

      height: 40vh;

      border: 5px solid red;

    }

  </style>

const grandParent = document.querySelector(".grandParent")

const parent = document.querySelector(".parent")

const child = document.querySelector(".child")

grandParent.addEventListener("click",()=>{

  alert("Grand parent")

},false)

parent.addEventListener("click",()=>{

  alert(" parent")

},false)

child.addEventListener("click",(e)=>{

  e.stopPropagation()

  alert("child")

},false)

1. What are the differences between '==' and '===' in JavaScript? Provide examples.

console.log(5 == "5"); // true, because the '==' operator converts the string "5" to a number and then compares 5 with 5.

console.log(true == 1); // true, because the boolean true is coerced to the number 1 and then compared to 1.

console.log(null == undefined); // true, because they are considered equal in loose comparison.

console.log(5 === "5"); // false, because the data types (number and string) are different.

console.log(true === 1); // false, because the data types (boolean and number) are different.

console.log(null === undefined); // false, because the data types (null and undefined) are different.

1. What is the purpose of the call(), apply(),'bind()' method in JavaScript? Provide an example.

🡺Simple example

* In call you can put arguments list
* In apply you can put argument Array (if array have two index then there are must be two parameters in function)
* In bind you can store the result into the variable (Don’t use array as an argument)

//call, bind, apply are the same

//If we borrowing function method then we are dependent to the object like

const obj ={

  name:"Adil younas",

  getName:function(){

    console.log(this.name);

  }

}

obj.getName() //Adil younas

const obj2 = {

  name:"Aqil younas"

}

obj.getName.call(obj2)  //Aqil younas

// {<------------------------>}

// In the above example we are dependent to the object but we want to access the funciton object independent

const obj = {

  name:"Adil younas"

}

function getName(){

  console.log(this.name);

}

getName.call(obj) //Adil younas

🡺apply() and bind()

//While using apply

function greet(name,age,location) {

  console.log(`${this.title}  ${name} ${age} ${location}`);

}

const person = { title: "Mr." };

const args = ["John",26,"Gujranwala"];

greet.apply(person, args); // Mr.  John 26 Gujranwala

// {<---------------------------->}

//how to use bind() no array use

const result = greet.bind(person,"John",27,"Gujranwala")

result()  //Mr.  John 27 Gujranwala

1. What are the differences between 'map()', 'filter()', and 'reduce()' array methods in JavaScript? Provide examples of their usage.

* map() transforms each element of the array and returns a new array of the same length.
* filter() creates a new array containing only the elements that pass a specific condition.
* Reduce() array into a single value, often used for calculations like summing or finding averages.

🡺map() after return use calculation

const arr = [1,2,3,4,5]

const result = arr.map((num)=>{

  return num \* num  //return array

})

console.log(result); //[1, 4, 9, 16, 25]

🡺filter() after return use condition

const students = [

  { name: "Alice", age: 25, grade: "A" },

  { name: "Bob", age: 22, grade: "B" },

  { name: "Charlie", age: 28, grade: "A" },

  { name: "David", age: 21, grade: "C" },

  { name: "Eva", age: 23, grade: "B" }

];

const result = students.filter((num)=>{

  return num.age>22

})

console.log(result); //[{name: 'Alice', age: 25, grade: 'A'},{name: 'Charlie', age: 28, grade: 'A'},{name: 'Eva', age: 23, grade: 'B'}]

🡺reduce() after return use

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

const sum = numbers.reduce((acc, num) => {

  return acc + num;

}, 0);

const average = sum / numbers.length;

console.log(sum); //15

console.log(average); //3

1. What is the difference between synchronous and asynchronous programming in JavaScript?

* Synchronous Programming: In synchronous programming, tasks are executed one after another. The program waits for each operation to finish before proceeding to the next line of code.
* Asynchronous Programming: In asynchronous programming, tasks can be started but not necessarily completed immediately. The program doesn't wait for the completion of a task before moving on to the next one.

console.log("Start");

console.log("Step 1");

console.log("Step 2");

console.log("End");

// Start

// Step 1

// Step 2

// End

console.log("Step1");

console.log("Step2");

setTimeout(() => {

  console.log("SetTime out");

}, 3000);

console.log("Step3");

// Step 1

// Step 2

// Step 3

// SetTime out

1. How do you handle JSON data in JavaScript? Explain the methods for parsing and stringifying JSON.

🡺JSON & BSON

* JSON is simple to read but make website slow
* BSON is 01 form but computer read it fast

const obj = {

  name:"adil",

  age:26

}

//object to json

const objectToJson = JSON.stringify(obj)

console.log(objectToJson); //{"name":"adil","age":26}

// {<-------------------->}

// Json to object

const jsonToObject = JSON.parse(objectToJson)

console.log(jsonToObject); //{name: 'adil', age: 26}

1. Explain the concept of destructuring assignment in JavaScript and provide an example.

//# OBJECT DESTRUCTURING

const obj = {

  name: "adil",

  age: 45,

  education: "msc",

};

const {name,age,education} = obj

console.log(name,age,education); //adil 45 msc

// {<--------------------------->}

//#ARRAY DESTRUCTURING

const arr = ["adil",45,"msc"]

const [name,age,education] = arr

console.log(name,age,education); //adil 45 msc

1. What are the differences between a shallow copy and a deep copy of an object or array in JavaScript?

🡺Shallow copy (not for nested objects or arrays)

* “A shallow copy is nothing but a clone of array or object but there is a catch that if there is an array inside an array / object in object then values of parent are copied but for child reference is copied”
* It is not perfect clone
* Nested arrays or objects are not copy as value they copy as a reference

//shallow copy of an array

const arr = [1, 2, [5, 8]];

const arr2 = [...arr];

arr2[2][0] = 1000;

console.log(arr); //[1,2,[1000,8]]

console.log(arr2); //[1,2,[1000,8]]

//#Shallow copy of object

const obj = {

  name:"adil",

  list:{

    name:"hamid",

    class:14,

    age:12

  }

}

const shallowCopy = {...obj}

shallowCopy.list.name = "Asad"

console.log(obj);  //{name:"adil",list:{name:"Asad",class:14 age:12}}

console.log(shallowCopy);  //{name:"adil",list:{name:"Asad",class:14 age:12}}

🡺deep copy (for nested objects or arrays)

//deep copy of an object

const obj = {

  name:"adil",

  list:{

    name:"hamid",

    class:14,

    age:12

  }

}

const deepCopy = JSON.parse(JSON.stringify(obj))

deepCopy.list.name = "Asad"

console.log(obj);  //{name:"adil",list:{name:"hamid",class:14 age:12}}

console.log(deepCopy);  //{name:"adil",list:{name:"Asad",class:14 age:12}}

//deep copy of an array

const arr = ["adil", 45, [10, 30]];

const deepCopy = JSON.parse(JSON.stringify(arr));

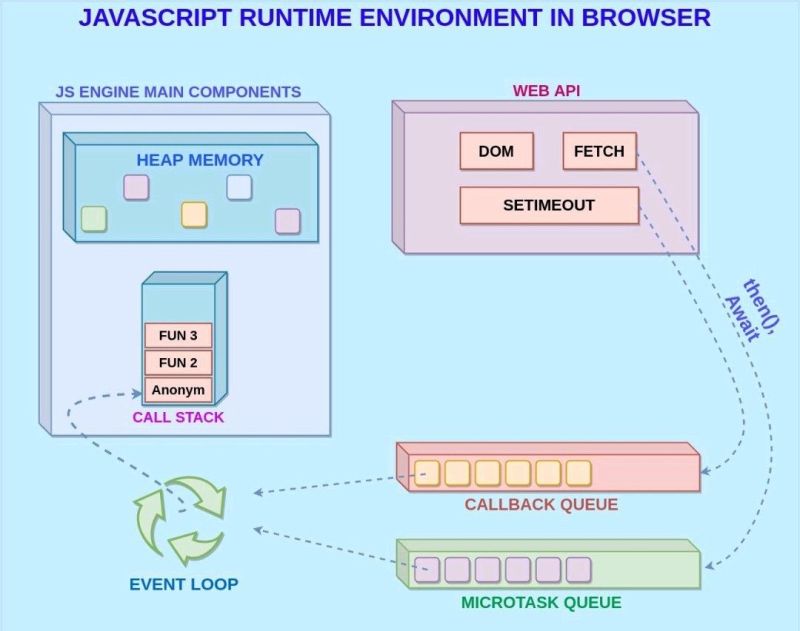
deepCopy[2][0] = 10000000000;

console.log(arr); //["adil",45,[10,30]]

console.log(deepCopy); //["adil",45,[10000000000,30]]

1. Explain the concept of the event loop in JavaScript and how it enables non-blocking operations.

This is very simple concept in which web an API pi who take time run its own environment and when data if received by API it will through the data into the message queue and event loop through it to the global execution context where data run



1. What is the difference between 'localStorage' and 'sessionStorage' in JavaScript? Explain their usage and limitations.

localStorage

**Scope**: Data stored in localStorage is available across all tabs and windows

**Lifetime**: The data stored in localStorage persists even after the browser is closed.

**Limitations**: Storing large amounts of data in localStorage may cause performance issues.

// Storing an object in localStorage

const user = { name: 'John', age: 30, email: 'john@example.com' };

localStorage.setItem('user', JSON.stringify(user));

// Retrieving the object from localStorage

const storedUser = JSON.parse(localStorage.getItem('user'));

console.log(storedUser); // Output: { name: 'John', age: 30, email: 'john@example.com' }

sessionStorage

**Scope**: Data stored in sessionStorage is available only within the same tab or window from which it was set

**Lifetime**: When the tab or window is closed, the data is cleared, and it is not available in subsequent sessions.

// Storing an array in sessionStorage

const shoppingCart = ['item1', 'item2', 'item3'];

sessionStorage.setItem('cart', JSON.stringify(shoppingCart));

// Retrieving the array from sessionStorage

const storedCart = JSON.parse(sessionStorage.getItem('cart'));

console.log(storedCart); // Output: ['item1', 'item2', 'item3']

1. What are the different methods for looping over arrays in JavaScript? Compare their usage and performance.

* Every loop is different via syntax so the performance is same in all as map() return new array so that it become a slight slow and for in loop is for object iteration so that it is not recommended for the object.

For loop

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

for (let index = 0; index < arr.length; index++) {

  console.log(arr[index]);

}

forEach() method

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

arr.forEach((element) => {

  console.log(element);

});

for of

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

for (const element of arr) {

  console.log(element);

}

map() method

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

const newArr = arr.map((element, index) => {

  return element \* index;

});

console.log(newArr);  //[0, 2, 6, 12, 20]

for in loop

const obj = {

  name: "adil younas",

  age: 45,

  location: "lahore",

};

for (const element in obj) {

  console.log(element);

}

1. Explain the concept of Recursion?

//Recursion: When function call itself

//otherwise: it will get called for infinite time

//if a function call itself then there must be an end point

//example of recursion but this will call itself infinite time

function recur() {

  console.log("You will got it");

  recur();

}

recur();

//In this example we use a condition to avoid the infinite loop

let counter = 1;

function demo(number) {

  if (counter > number) {

    return;

  }

  console.log(counter);

  counter++;

  demo(number);

}

demo(10);

1. "Explain the use and benefits of the rest and spread operators in JavaScript. Provide examples of how each operator can be used to manipulate arrays and function arguments effectively."

Spread Operator (...)

Example of using the spread operator to merge arrays

const arr1 = [1, 2, 3];

const arr2 = [4, 5, 6];

const mergedArray = [...arr1, ...arr2];

console.log(mergedArray); // Output: [1, 2, 3, 4, 5, 6]

Example of using the spread operator with function arguments

function greet(name, age, city) {

  console.log(`Hello, ${name}! You are ${age} years old and live in ${city}.`);

// Output: Hello, Alice! You are 30 years old and live in New York.

}

const userDetails = ["Alice", 30, "New York"];

greet(...userDetails);

Rest Operator (...)

* When you are receiving …args in function parameters then it is rest in simple terms

function sum(...numbers) {

    return numbers.reduce((acc, num) => acc + num, 0);

  }

  const result = sum(1, 2, 3, 4, 5); // The rest operator captures all arguments into the 'numbers' array.

  console.log(result); // Output: 15

**Array Methods**

toString() & join() Change Existing Array

const arr = [1,2,3,4,5,6,7,8,9]

const result = arr.toString()

console.log(result); //whole array now a sting

console.log(typeof result); //String

const arr2 = [1,2,3,4,5,6,7,8,9]

const result2 = arr2.join("\*")

console.log(result2); //elements are separated by \* (1\*2\*3\*4\*5\*6\*7\*8\*9)

console.log(typeof result2); //string

Sort(), reverse() Change existing Array

//shorting of array has elements Strings

const stringArr = ["adil", "younas", "rizwan", "zaman"];

stringArr.sort(); //A-Z formate

console.log(stringArr);

stringArr.sort().reverse(); //Z-A formate

console.log(stringArr);

//sorting for number array

const numberArr = [5, 3, 7, 8, 1];

numberArr.sort((a, b) => a - b);

console.log(numberArr); //[1, 3, 5, 7, 8]

numberArr.sort((a, b) => b - a);

console.log(numberArr); //[8, 7, 5, 3, 1]

push(), pop(), shift(), unshift() Change Existing Array

// push(), pop(),  unshift()  ,shift(),

//push() add in last

const arr = [14,25,21]

arr.push(100)

console.log(arr); //[14, 25, 21, 100]

//pop() delete from last

const arr2 = [58,62,75]

arr2.pop()

console.log(arr2); //[58, 62]

//unshift() add at [0] index

const unShiftArr = [1,8,452,45,4,521]

unShiftArr.unshift("adil")

console.log(unShiftArr); //['adil', 1, 8, 452, 45, 4, 521]

//unshift() add at [0] index

const shiftArr = ["adil",1,8,452,45,4,521]

shiftArr.shift("adil")

console.log(shiftArr); // [1, 8, 452, 45, 4, 521]

Splice() Change Existing Array

* Both splice and slice cut a piece of cake so do not start counting index from [0]. Both count from sides of index in this case left side of 0

const ages = [33, 12, 20, 16, 5];

ages.splice(0, 1, "adil");

console.log(ages); //['adil', 12, 20, 16, 5]

Slice() Create a New Array

//if you didn’t store it into the variable then it will result different

const ages = [33, 12, 20, 16, 5];

const result = ages.slice(0,3)

console.log(result); //[33, 12, 20]

Concat() Create a New Array

const array1 = [1, 2];

const array2 = [3, 4];

const array3 = [5, 6];

const combinedArray = array1.concat(array2, array3);

console.log(combinedArray); // Output: [1, 2, 3, 4, 5, 6]

console.log(array1); // Original array remains unchanged: [1, 2]

console.log(array2); // Original array remains unchanged: [3, 4]

console.log(array3); // Original array remains unchanged: [5, 6]

Includes () & indexOf ()

//includes()

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

console.log(numbers.includes(3)); // Output: true

console.log(numbers.includes(6)); // Output: false

//indexOf()

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

console.log(fruits.indexOf('banana')); // Output: 1

console.log(fruits.indexOf('watermelon')); // Output: -1

Find () & filter ()

* Find run until the first match is satisfied
* Filter run on whole array

let arrray = [25, 30, 45, 66, 88]

let data = arrray.find((val) => {

return val > 25

})

console.log(data, typeof data); //30 'number'

let data2 = arrray.filter((val) => {

return val > 25

})

console.log(data2, typeof data2); //[30, 45, 66, 88] 'object'

**OOP (object-oriented programming)**

Factory functions

* Simple function return object which is use to scale up the project
* It use return not new

function createPerson(name, age) {

    return {

      name: name,

      age: age,

      sayHello: function() {

     console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);

      }

    };

  }

  const person1 = createPerson('Alice', 30);

  const person2 = createPerson('Bob', 25);

  person1.sayHello(); // Output: Hello, my name is Alice and I am 30 years old.

  person2.sayHello(); // Output: Hello, my name is Bob and I am 25 years old.

Constructor function

* Constructor functions create reusable object blueprints.
* It uses new to return
* Use can use constructor that is why we call it constructor function

function Person(name, age) {

  this.name = name;

  this.age = age;

  this.sayHello = function () {

    console.log(

      `Hello, my name is ${this.name} and I am ${this.age} years old.`

    );

  };

}

const person1 = new Person("Alice", 30);

console.log(person1);

 console.log(person1.constructor);

Class (Use for Inheritance)

* In JavaScript, a class is a blueprint for creating objects with specific properties and behaviors.
* Introduced in ECMAScript 6 (ES6), the class syntax is a syntactical sugar over the existing prototype-based inheritance in JavaScript because JS Is prototype-based inheritance language not OOPs. OOPs are used for other developers who actually are using OOPS in other language like JAVA. It is easy for them to use JS OOPS concepts. It is a sugar cotted thing in JS
* You can access private variables using in any method in class.

class Family {

  #familySystem;

  constructor(mName, fName, kids, familySystem) {

    this.mName = mName;

    this.fName = fName;

    this.kids = kids;

    this.#familySystem = familySystem;

  }

  print(){

    console.log("I am printing");

  }

}

const parents = new Family("younas", "shafqat", 3, "Joint");

console.log(parents);

class Me extends Family {

  constructor(mName, fName, kids, familySystem, location) {

    super(mName, fName, kids, familySystem); //parent class constructor so feed it

    this.location = location;

  }

}

const me = new Me("adil", "Ayesha", 2, "Nuclear", "Lahore");

console.log(me);

me.print()

# React JS

Basic React JS Questions

1. What is React JS?

* Open-source JavaScript library
* Developed and maintain by Facebook
* Reusable UI Components
* It uses virtual DOM
* It supports both client-side rendering and server-side rendering
* You can combine react with libraries

1. What are the features and advantages of React JS?

* It uses virtual DOM: it is a lightweight copy of the actual DOM.
* Component based Architecture: where UI is broken down into small and reusable components.
* JSX (JavaScript XML): syntax like HTML it is the combination of HTML + JavaScript
* Unidirectional: data flow or event delegation in same direction parent to child it means event capturing.
* React JS and React natives are most similar so if a person can code in react JS then he definitely can code in react native.
* Community and Ecosystem: React has a large and active community, which means a vast number of third-party libraries, tools and resources are available.

1. What is JSX?

JSX (JavaScript XML): syntax like HTML it is the combination of HTML + JavaScript

1. What is the difference between a component and an element?

🡺Component (Parent level)

// Example of a React component (function component)

const GreetingComponent = (props) => {

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

};

🡺Element (Child level)

// Example of a React element created using the GreetingComponent

const element = <GreetingComponent name="John" />;

export default element;

1. Explain the controlled and uncontrolled components in react JS? Or what are the types of components?

**Majorly there are two types of components**

* Controlled component
* Uncontrolled component

Controlled Component:

A controlled component is a component in which the value of the form element is controlled by React's state.

import React, { useState } from 'react';

function ControlledInput() {

  const [inputValue, setInputValue] = useState('');

  const handleInputChange = (event) => {

    setInputValue(event.target.value);

  };

  return (

    <input

      type="text"

      value={inputValue}

      onChange={handleInputChange}

    />

  );

}

Uncontrolled Component:

An uncontrolled component is a component where the value of the form element is handled by the DOM itself, rather than being controlled by React's state. The value of the input is accessed using a ref, and changes to the input value are not managed through state updates.

import React, { useRef } from 'react';

function UncontrolledInput() {

  const inputRef = useRef(null);

  const handleButtonClick = () => {

    console.log('Input value:', inputRef.current.value);

  };

  return (

    <div>

      <input type="text" ref={inputRef} />

      <button onClick={handleButtonClick}>Log Value</button>

    </div>

  );

}

**Which to Use:**

Controlled components are generally recommended in React because they provide a clear and predictable way to manage and validate form input. Controlled components allow you to validate and manipulate the data before it's used, and they also fit well with React's data flow model.

1. How do you create components in React JS?

“In React JS, there are two main ways to create components: function components and class components.”

**Function Components: (constructor function)**

Function components are simpler and considered the modern way of creating components in React. They are JavaScript functions that return JSX (React elements). Function components are used for stateless UI components, which receive data via props and display it.

import React from 'react';

// Function component example

const MyComponent = (props) => {

  return (

    <div>

      <h1>Hello, {props.name}!</h1>

      <p>{props.message}</p>

    </div>

  );

};

export default MyComponent;

**Class Components: (constructor function):**

“Class components are an older approach to creating components and are still widely used, especially when you need to manage state or use lifecycle methods.”

import React from 'react';

// Class component example

class MyComponent extends React.Component {

  render() {

    return (

      <div>

        <h1>Hello, {this.props.name}!</h1>

        <p>{this.props.message}</p>

      </div>

    );

  }

}

export default MyComponent;

State and Props Questions

1. What is the difference between state and props?

🡺Props

Props in React are like messages from a parent to a child component. They provide data for customization, but the child can't change the message – it's read-only. This ensures a clear flow of information from parent to child.

import React from 'react';

const ParentComponent = () => {

  const name = "John";

  return <ChildComponent name={name} />;

};

const ChildComponent = (props) => {

  return <p>Hello, {props.name}!</p>;

};

🡺State

In React, "state" refers to the data that a component can hold and manage. It represents the information that can change over time and influences what gets displayed in the user interface. Components can read and update their own state, and when state changes, React automatically re-renders the component to reflect those changes.

import React, { useState } from 'react';

const Counter = () => {

  // Using the useState hook to define state

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

  const handleIncrement = () => {

    // Updating the state using setCount

    setCount(count + 1);

  };

  const handleDecrement = () => {

    // Updating the state using setCount

    setCount(count - 1);

  };

  return (

    <div>

      <p>Count: {count}</p>

      <button onClick={handleIncrement}>Increment</button>

      <button onClick={handleDecrement}>Decrement</button>

    </div>

  );

};

export default Counter;

“In summary, props are used to pass data from parent to child components, and they are immutable. State, on the other hand, is managed internally within a component and can be changed, triggering re-renders of the component with the updated data.”

1. How to pass parameter through event handler?

import React from 'react';

function Component() {

  const handleClick = (param) => {

    console.log('Button clicked with parameter:', param);

  };

  return (

    <button onClick={() => handleClick('Hello')}>

      Click me

    </button>

  );

}

export default Component;

1. What is lifting state up in react JS?

* Lifting state of means empowering child component with data and functions
* In react JS data flows uni-directional it means that data travel parent to child. The prop we pass child is read only, but in this lifting state up we give function and setstate() to the child so that the child will also change the state and trigger the function

import React, { useState } from 'react';

function ParentComponent() {

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

  const incrementCount = () => {

    setCount(count + 1);

  };

  return (

    <div>

      <ChildComponent count={count} onIncrement={incrementCount} />

      <ChildComponent count={count} onIncrement={incrementCount} />

    </div>

  );

}

function ChildComponent({ count, onIncrement }) {

  return (

    <div>

      <p>Count: {count}</p>

      <button onClick={onIncrement}>Increment</button>

    </div>

  );

}

1. What are hooks are why we use them?

* Hooks in React are special functions that allow you to "hook into" React state and lifecycle features from functional components, which traditionally only class components could access. Hooks were introduced to simplify and enhance the way stateful logic is managed and reused in React applications.
* For functional component
* Use to manage state
* Use for state logics

1. What are some of the best practices for managing state in React JS?

* Use Functional components
* One component in one file
* Every variable and function should have proper name according to its purpose
* use Redux for state managements

Virtual DOM Questions

1. What is the virtual DOM?

* Make Client-side rendering smooth
* However, directly updating the real DOM can be computationally expensive, especially when dealing with large or complex user interfaces. Every time there is a change in the application state that affects the UI, the browser needs to recalculate the entire DOM tree and apply the changes, even for minor updates. This process is time-consuming and can lead to performance bottlenecks, especially on devices with limited resources.
* The Virtual DOM is a lightweight copy of the real DOM, maintained by React in memory. When there is a change in the application state, React uses a process called "reconciliation" to calculate the difference between the previous Virtual DOM and the updated Virtual DOM (after the state change). This difference is known as the "diff."
* By comparing the differences in the Virtual DOM, React efficiently identifies the minimal number of changes required to update the actual DOM. It then applies these changes directly to the real DOM in a single batch, minimizing expensive DOM operations.

1. Explain the term “reconciliation” and “diff” and “React Fiber?”

**Reconciliation: (Process)**

“Reconciliation is the process by which React compares the previous Virtual DOM representation of a component with its updated Virtual DOM representation after a state change or other updates. The goal of reconciliation is to determine the minimal number of changes needed 2. Diff (Difference)

**Diff: (result of reconciliation to identify difference)**

The "diff" is the result of the reconciliation process—the set of changes required to transform the previous Virtual DOM into the updated Virtual DOM. The diffing algorithm efficiently identifies which elements or components have changed, which ones need to be added or removed, and which ones can stay the same.to update the actual DOM and reflect the changes in the user interface.”

**React Fiber: (fiber by fiber change to prevent CPU blockage)**

Think of React Fiber like a smart task scheduler for a CPU. Just as a CPU needs to efficiently manage various tasks to keep a computer responsive, React Fiber efficiently manages updates to a user interface. It breaks down UI updates into smaller tasks, organizes them by priority, and can pause and resume tasks as needed. This way, it ensures that the user interface remains smooth and responsive, similar to how a well-managed CPU ensures your computer doesn't slow down even when handling multiple tasks.

1. Explain server-side rendering and client-side rendering

**Server-Side rendering:**

* In server-side rendering server initially process the code and create a HTML page and give it to the browser and the browser render the HTML page after that browser download the react code and JS and apply it to HTM. Server-side rendering say to the browser to show initial HTML (layout for client side) then I will give you the rest of the code which consist logics etc. In this way the first render will become easy.
* Server-side rendering (SSR) is a technique used in web development to render the initial HTML content of a web page on the server side before sending it to the client's browser. This approach is used to improve factors like search engine optimization (SEO), initial page load times, and accessibility.

**Client-Side rendering:**

* In client side rendering a bundle is send to the browser and browser will separate the HTLM file logics and other things
* This process is time taking and use browser and system resources, this way the first render will take time.

1. What are the Synthetic events?

* As events are logics, logic can be written in programming language like JS, and V8 engine can understand the JS, and browser hold the V8 engine so, that events are the part of browsers.
* Events handling is a part of browser but what if you are using different browser which does not support and particular event. To solve this problem, react deal with events understand the event and after that tell the browser that user is talking about this particular event. This ensures the same result in all browsers. This feature also presents in jQuery.