Top javascript questions for interview:

Understanding the Basics of JavaScript

JavaScript, created by Brendan Eich in 1995, has grown to become a cornerstone of modern web development. It is essential for creating interactive web pages and is supported by all major browsers. Its versatility extends from client-side to server-side development, thanks to environments like Node.js. Understanding JavaScript's basics is crucial for any aspiring web developer, as it lays the foundation for more advanced concepts and frameworks.

What are variables in Javascript?

One of the first concepts to grasp is variables. In JavaScript, variables can be declared using var, let, or const. While var has function scope, let and const provide block-level scope, with const being immutable after its initial assignment. For example:

```
let name = 'John';

const age = 30;
```

What are the data types in Javascript?

JavaScript supports several data types including string, number, boolean, null, undefined, and object. Understanding these types and their behaviors is vital. For instance, arithmetic operations on numbers are straightforward, but attempting arithmetic with a string can lead to unexpected results due to type coercion.

Javascript questions for interview – Interview Preparation support

If you are preparing for Javascript interview. we can help you to prepare for the interview. you can connect with us on whatsupp +91-8527854783

What is Function in Javascript?

Functions in JavaScript is a piece of code implemented to perform a particular task. Function keyword or as a fat arrow could be used to

declare the functions. Functions help in code reusability and modularity. For example:

```
function greet(name) {
  return `Hello, ${name}!`;
}
console.log(greet('Alice'));
```

What is closure in Javascript?

A closure in JavaScript is a function that retains access to its lexical scope even when the function is executed outside that scope. Closures are often used to create private variables or functions, making them a powerful tool for implementing encapsulation. For example:

```
function outerFunction() {
  let privateVariable = 'Javascript questions for interview article - Definition hoisted!';
  return function innerFunction() {
    console.log(privateVariable);
  };
}
const myFunction = outerFunction();
myFunction(); // Outputs: I am private
```

What is Generators in Javascript?

It’s a function/ iteration function that we defined by using * infront of function declaration which returns the Generator object and this object has the next() iteration function

- In Generator function, we use yield to pause and resume it which returns an object having two properties. One is value which is the actual value and the second is done – it’s a boolean if it’s true that means Generation function is fully completed otherwise it would return a false
- We use Generation function to control the iteration
- The main difference between iteration functions like for-loop and Generator is that if we execute for loop it will execute for all iterations immediately where as Generator can

pause and resume its iteration and whenever we want a next value we use next() iteration function of Generator object

```
function* genertorForLoop(num) {

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

    yield console.log(i)

  }

}

var i = genertorForLoop(5)

i.next().value
```

What is Asynchronous programming in JavaScript?

Asynchronous programming in JavaScript is crucial for performing operations like API calls without blocking the execution thread. Promises are a common way to handle asynchronous operations, providing a more readable and maintainable approach compared to traditional callbacks. A promise is a function that promises that it will be resolved/rejected now or in the future. Here is an example:

```
let promise = new Promise((resolve, reject) => {

  setTimeout(() => {

    resolve('Data fetched successfully - Javascript questions for interview article - Definition hoisted!');

  }, 2000);

});

promise.then(result => {

  console.log(result); // Outputs: Data fetched successfully

});
```

What is Event loop?

The event loop is another critical concept in JavaScript, ensuring non-blocking I/O operations. It allows the execution of multiple operations by managing the call stack and task queue. Understanding how the event loop

works can help in optimizing code and preventing common pitfalls like race conditions.

What is debouncing and throttling in Javascript?

When addressing common JavaScript problems, such as debouncing and throttling, it is important to understand their impact on performance. Debouncing ensures that a function is not called too frequently, whereas throttling limits the number of times a function can be executed over time.

Throttling is one of a kind of writing code in which, it does not matter, how many times, you fire the event, the attached function will be executed only once in a given time interval.
In the debouncing, it does not matter how many times you fire *the event, the attached function will be executed only after the specified time once the user stops firing the event.*
Here's a basic debounce function:

```
function debouncing(fun, delay) {
  clearTimeout(timeid);
  timeid = setTimeout(function () {
    fun()
  }, delay)
}
```

Below is a throttling example

```
function f1() {
}


function throtling() {
  if (timeid) {
    return;
  }
```

```
 var timeid = setTimeout(function () {
  f1();
  timeid = undefined;
 }, delay)
}
```

What is Async / Await and Promise in Javascript?

- The async function returns a promise.
- Await blocks the code of execution with in async function and it only makes sure that the next line will be executed once the promise is resolved or rejected
- To run multiple promises parallel, we used the promise.all([P1, P2, P3,...etc]), and if one of them is rejected then promise.all will be rejected
- To avoid Promise.all rejection scenarios in case of one promise fail, we can map function and can return undefined / error message in case any promise gets failed

```
Promise.all([p1, p2, p3,..pn].map(p => p.catch(err => err))
```

How do we do deepClone of an object in Javascript?

It’s a function that you can use to get a deep copy of an object. Please see the example below for the deep clone

```
function deepClone(obj) {
  var clone = {};
 for( let i in obj) {
  if(typeof(obj[i]) ==="object" && obj[i] !== null) {
    clone[i] = deepClone(obj[i]);
  } else {
    clone[i] = obj[i];
  }
 }
 return clone;
```

```
}
```

Write polyfills for Object. assign in JavaScript

```
function objectAssign(srcObj, targetObj) {


    let obj = {};
    let allKeys = new Set([...Object.keys(srcObj), ...Object.keys(targetObj)])
    allKeys.forEach((key) => {
        console.log(key, Array.isArray(srcObj[key]))
        if (Array.isArray(srcObj[key]) && Array.isArray(targetObj[key])) {
            obj[key] = srcObj[key].concat(targetObj[key]);
        } else {
            obj[key] = targetObj[key] || srcObj[key];
        }
    })
    return obj;
}
```

Advanced javascript questions for interview – Interview Preparation support

If you are preparing for Javascript interview. we can help you to prepare for interview. you can connect with us on whatsupp [+91-8527854783](tel:+91-8527854783)

Write a program to find missing numbers in an array using Javascript

```
function findMissing(arr) {
    var max = Math.max(...arr);
    var min = Math.min(...arr);
    var missing = [];
    for (var i = min; i <= max; i++) {
```

```
        if (!arr.includes(i)) {
            missing.push(i);
        }
    }
    return missing.join(",");
}
var arr = [1, 2, 5];

findMissing(arr);
```

Output: 3, 4

Write pollyfills for bind in JavaScript

```
let obj = {
    a: 32,
    getA: function () {
        return this.a;
    }
}

Function.prototype.cbind = function (ref) {
    if (typeof (this) !== 'function') {
        throw new Error("Its not function");
    }

    let func = this;
```

```
    return function () {
        return func.apply(ref);
    }
}

let func1 = obj.getA;

let func = func1.cbind(obj)


console.log(func())
```

What is a Prototype in Javascript?

The prototype is a property of a JavaScript function. Each time we create a function in JavaScript,
JavaScript engine automatically attached an extra property called prototype to the created function. This prototype property is an object that itself has a constructor property by default. This constructor property points back to the function object on which the prototype object is a property

What is Factory function in Javascript?

A factory function is a function that is not a class or constructor that returns an object. In JavaScript, any function can return an object. When it does so without the new keyword, it's a factory function.

```
function person(firstName, lastName, age) {
    const person = {};
    person.firstName = firstName;
    person.lastName = lastName;
    person.age = age;
    return person;
}
```

What is Prototype chain in Javascript?

All the objects in JavaScript inherit the properties and methods from Object.prototype. This is called Prototype chaining.

What is __proto__ in Javascript?

The **__proto__** property is an accessor property on object consisting getter and setter method

What is the difference b/w __proto__ and prototype?

Prototype is a property belonging only to functions. It is used to build **__proto__** Where **__proto__ is a property of an** object consisting getter and setter method

How we can extend from the parent class method to the child class in Javascript?

```
function Teacher(name) {
   this.name = name;
}
function Student(sname) {
   this.sname = sname;
}
Student.prototype = new Teacher("Bob");
Student.prototype.showMsg = function () {
   console.log(this.sname + "'s teacher name is " + this.name)
}
var s1 = new Student("Pradeep");
s1.showMsg()
```

What is Event delegation in Javascript?

Event delegation is where we add event listeners to the top element(parent element) instead of adding them to the child elements and

once you click on the child element then your event would be captured with the event handler attached to the parent element.

```
<div id='div'>
  <button id='button-a'>
    Click Me
  </button>
  <button id='button-b'>
    Click Me
  </button>
</div>


const div = document.getElementById('div');
div.onclick = (event) => {
  if (event.target.matches("button#button-a")) {
    alert('Button A clicked');
  } else if (event.target.matches("button#button-b")) {
    alert('Button B clicked');
  }
}
```

What is Hoisting in Javascript?

Hoisting is a mechanism in JavaScript where function and variable declarations are automatically moved to the top of their scope before code execution. In the case of a function, a function declaration doesn't just hoist the function's name. It also hoists the actual function definition as well. Below is one of example for function hoisting

```
 // Outputs: "Javascript questions for interview article - Definition hoisted!"
```

```
definitionHoisted();


// TypeError: undefined is not a function
definitionNotHoisted();


function definitionHoisted() {
    console.log("Javascript questions for interview article - Definition hoisted!");
}


var definitionNotHoisted = function () {
    console.log(" Javascript questions for interview article - Definition not hoisted!");
};
```

What is TypeScript?

TypeScript is a syntax version of JavaScript with type support.

Advantages of using typescript:

- It comes with ECMAScript which means you can use const, let, rest, etc while you are coding
- It supports static typing
- It gives better IDE support

Disadvantage: You can not run Typescript directly in the browser. A transpiler would be required to transform Typescript into Javascript then it could be useful for browsers.

Write a program to check balanced parenthesis

```
var str = "{([])}";
function checkForBalancedParenthesis(str) {
    var arr = [];
    var obj = {
```

```
        "{": "}",
        "(": ")",
        "[": "]"
    }

    var s = str.split("");
    for (var i = 0; i < s.length; i++) {
        if (s[i] === "{" || s[i] === "[" || s[i] === "(") {
            arr.push(s[i]);
        } else {
            var lastelem = arr.pop();
            if (s[i] !== obj[lastelem]) {
                return false;
            }
        }
    }

    return true;

}

console.log(checkForBalancedParenthesis(str))
```

Output - true

Write a program to find the valid longest parenthesis

```
var s = ")()())";
```

```
function findValidParenthisis(str) {
    var arr = [], left = 0, right = 0, max = 0;
    var s = str.split("");
    for (var i = 0; i < s.length; i++) {
        if (s[i] === "(") {
            left++;
        } else {
            right++;
        }

        if (left === right) {
            max = Math.max(max, 2 * right);
        } else if (right >= left) {
            left = right = 0;
        }
    }
    left = right = 0;
    for (j = s.length - 1; j >= 0; j--) {
        if (s[j] === "(") {
            left++;
        } else {
            right++;
        }
        if (left === right) {
            max = Math.max(max, left * 2)
```

```
    } else if (left >= right) {
        left = right = 0;
    }
  }
  return max;
}
```

```
console.log(findValidParenthisis(s));
```

Some useful links along with javascript questions for interview article.
Integrate React js with angular js – https://tekody.com/blogs/integrating-react-js-components-with-angular-js-directives/

Online React js support service from India:
https://tekody.com/services/react-js-job-support/

Javascript communities –https://dev.to/t/javascript