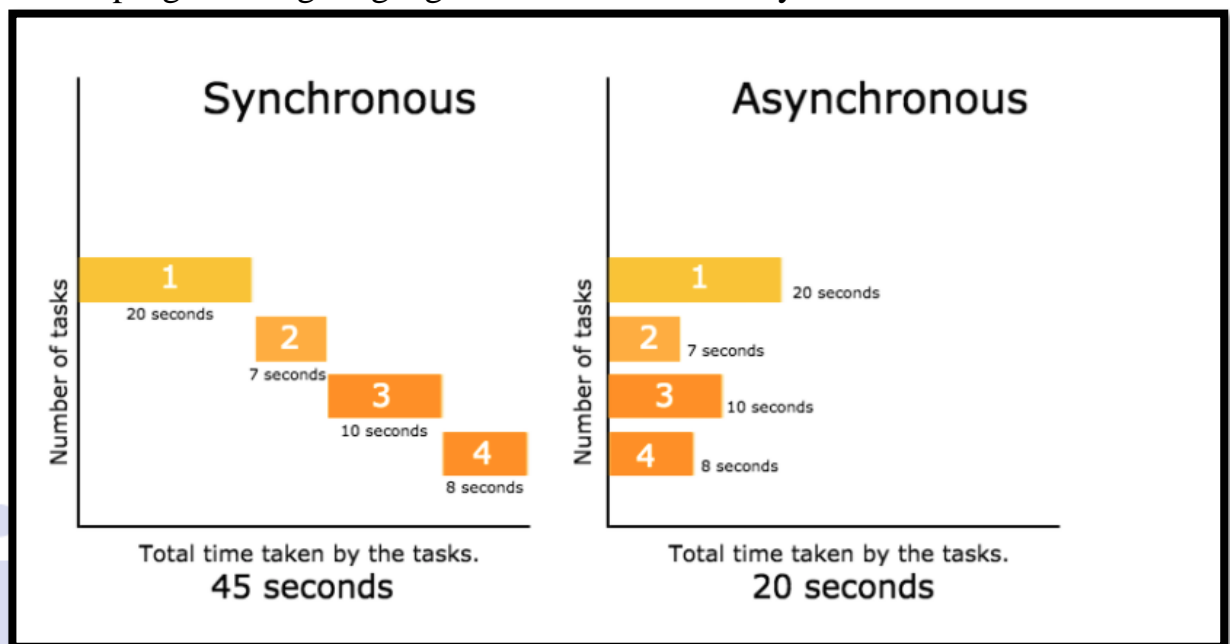




## INTERVIEW QUESTIONS

1. What is the Nature of JavaScript?

Ans. JavaScript is a single-threaded, non-blocking, asynchronous, concurrent programming language with lots of flexibility.



2. What are function scope and block scope?

Ans.

**Function Scope:** When a variable is declared inside a function, it is only accessible within that function and cannot be used outside that function.

```
JS index.js > ...
1  function ineuron(){
2      let message = "Welcome to iNeuron"
3      console.log(message) //Message is scoped to the ineuron function
4  }
5  console.log(message) // Not Defined
6  ineuron()
7
```



**Block Scope:** A variable when declared inside the if or switch conditions or inside for or while loops, are accessible within that particular condition or loop.

```
1 function ineuron(){
2     let message = "Welcome to iNeuron"
3     console.log(message) //Message is scoped to the ineuron function
4     if(true){
5         let message = "Welcome to Block Scope"
6         console.log(message)
7     }
8 }
9 ineuron()
```

To be concise the variables declared inside the curly braces are called as within block scope.

3. What is Synchronous and Asynchronous in JavaScript?

Ans.

### **Synchronous**

- Stops execution of further code until this is done.
- Because of this stoppage of further execution, synchronous code is called 'blocking'. Blocking in the sense that no other code will be executed.

### **Asynchronous**

- Execution of this is deferred to the event loop, this is a construct in a JS virtual machine which executes asynchronous functions (after the stack of synchronous functions is empty).
- Asynchronous code is called non-blocking because it doesn't block further code from running.



```
1 // This function is synchronous
2 function ineuron(msg) {
3     console.log(msg)
4 }
5
6 ineuron("Hello Neurons");
7
8 // This function is asynchronous
9 setTimeout(() => {
10     console.log("I will run after couple of seconds")
11 }, 100);
```

4. Is JavaScript a statically typed or a dynamically typed language.

Ans. JavaScript is a dynamically typed language, you can go about declaring variables, functions, objects and anything without declaring the type.

5. What is hoisting in JavaScript?

Ans. JavaScript Hoisting refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

Hoisting allows functions to be safely used in code before they are declared.

```
1 ineuron("Hitesh");
2
3 function ineuron(name) {
4     console.log("Welcome " + name);
5 }
```

Without Hoisting same thing will be declared as



```
1 function ineuron(name) {  
2   console.log("Welcome " + name);  
3 }  
4 ineuron("Hitesh");
```

6. What is undeclared and undefined in JavaScript.

Ans. Undefined variable means a variable has been declared but it does not have a value. Undeclared variable means that the variable does not exist in the program at all.

```
1 let message;  
2 console.log(message) //Undefined  
3 console.log(msg) //Undeclared
```

7. Is JavaScript is pass by value or pass by reference.

Ans. It's always pass by value, but for objects the value of the variable is a reference. Because of this, when you pass an object and change its members, those changes persist outside of the function. This makes it look like pass by reference. But if you actually change the value of the object variable you will see that the change does not persist, proving it's really pass by value.

8. What is "this" in JavaScript?

Ans. In JavaScript, the this keyword refers to an object.



- Which object depends on how this is being invoked (used or called).  
The “this” keyword refers to different objects depending on how it is used
- In an object method, this refers to the object.
  - Alone, this refers to the global object.
  - In a function, this refers to the global object.
  - In an event, this refers to the element that received the event.

```
1 var obj = {  
2   name: "iNeuron",  
3   getName: function(){  
4     console.log(this.name);  
5   }  
6 }  
7  
8 obj.getName();
```

9. What is arrow function and how to write it.

Ans. Arrow functions were introduced in the ES6 version of JavaScript. They provide us with a new and shorter syntax for declaring functions. Arrow functions can only be used as a function expression.

```
1 // Traditional Function Expression  
2 var add = function(a,b){  
3   return a + b;  
4 }  
5  
6 // Arrow Function Expression  
7 var arrowAdd = (a,b) => a + b;
```

Arrow functions are declared without the function keyword. If there is only one returning expression, then we don't need to use the return keyword.



Also, for functions having just one line of code, curly braces { } can be omitted.

#### 10. Difference between let, var and const

Ans. var variables can be updated and re-declared within its scope.

let variables can be updated but not re-declared.

const variables can neither be updated nor re-declared.

Keyword	const	let	var
Global Scope	No	No	Yes
Function Scope	Yes	Yes	Yes
Block Scope	Yes	Yes	Yes
Can be reassigned	No	Yes	Yes

#### 11. What is DOM and Virtual DOM?

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.

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM. This process is called reconciliation.

#### 12. Explain Closures in JavaScript.

Ans. A closure is the 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.

A closure is a pairing of:

- A function and
- A reference to that function's outer scope (lexical environment)

#### Lexical Scoping



A lexical scope in JavaScript means that a variable defined outside a function can be accessible inside another function defined after the variable declaration. But the opposite is not true; the variables defined inside a function will not be accessible outside that function.

```
1  var x = 2;
2  var add = function() {
3      var y = 1;
4      return x + y;
5  };
6  var lex = add()
7  console.log(lex)
```

### Closures

```
1  function assign() {
2      var name = 'iNeuron';
3
4      function displayName() {
5          console.log(name);
6      }
7      return displayName;
8  }
9
10 var myFunc = assign();
11 myFunc();
```

13. What is rest parameter and spread operator?

Ans. Both rest parameter and spread operator were introduced in the ES6 version of JavaScript.

### Rest Parameter

It provides an improved way of handling parameters of a function. Using the rest parameter syntax, we can create functions that can take a variable



number of arguments. Any number of arguments will be converted into an array using the rest parameter. It also helps in extracting all or some parts of the arguments.

Rest parameter can be used by applying three dots (...) before the parameters.

```
1 function addAllArgs(...args){
2     let sum = 0;
3     let i = 0;
4     while(i < args.length){
5         sum += args[i];
6         i++;
7     }
8     return sum;
9 }
10
11 console.log(addAllArgs(6, 5, 7, 99)); // Returns 117
12 console.log(addAllArgs(1, 3, 4)); // Returns 8
```

Spread Operator





Although the syntax of spread operator is exactly the same as the rest parameter, spread operator is used to spread an array, and object literals. We also use spread operators where one or more arguments are expected in a function call.

```
1  const mylap = {
2    brand: 'Apple',
3    model: 'Pro',
4    color: 'Silver'
5  }
6
7  const updatelap = {
8    type: 'Laptop',
9    year: 2022,
10   color: 'Space Grey'
11  }
12
13  const myupdatedlaptop = {...mylap, ...updatelap}
14  console.log(myupdatedlaptop)
```

#### Key Difference between rest operator and spread operator

- Rest parameter is used to take a variable number of arguments and turns into an array while the spread operator takes an array or an object and spreads it.
- Rest parameter is used in function declaration whereas the spread operator is used in function calls.

#### 14. What are promises in JavaScript and How to use of promises in JavaScript?

Ans. Promises are used to handle asynchronous operations in JavaScript. Before promises, callbacks were used to handle asynchronous operations. But due to limited functionality of callback, using multiple callbacks to handle asynchronous code can lead to unmanageable code.

Promise object has four states:



Pending - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.

Fulfilled - This state represents that the promise has been fulfilled, meaning the async operation is completed.

Rejected - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.

Settled - This state represents that the promise has been either rejected or fulfilled.

A promise is created using the Promise constructor which takes in a callback function with two parameters, resolve and reject respectively.

resolve is a function that will be called, when the async operation has been successfully completed.

reject is a function that will be called, when the async operation fails or if some error occurs.

Creating Promise



```
1 function sumOfThreeElements(...elements){
2   return new Promise((resolve,reject)=>{
3     if(elements.length > 3 ){
4       reject("Only three elements or less are allowed");
5     }
6     else{
7       let sum = 0;
8       let i = 0;
9       while(i < elements.length){
10        sum += elements[i];
11        i++;
12      }
13      resolve("Sum has been calculated: "+sum);
14    }
15  })
16 }
```

We can consume any promise by attaching then() and catch() methods to the consumer.

then() method is used to access the result when the promise is fulfilled.

catch() method is used to access the result/error when the promise is rejected.

### Consuming Promise

```
1 sumOfThreeElements(4, 5, 6)
2 .then(result=> console.log(result))
3 .catch(error=> console.log(error));
```

15. What is Scope Chain in JavaScript

Ans. Scope chains establish the scope for a given function. Each function defined has its own nested scope, and any function defined within another



function has a local scope which is linked to the outer function this link is called the chain.

```
1 function parent() {
2   var name = 'Aastha';
3   console.log(name);
4   console.log(age); // Reference error: age is not defined
5   console.log(places); // Reference error: places is not defined
6
7   function child() {
8     // function linked to parent() thats why name is accessible.
9     var age = 23;
10    console.log(name);
11    console.log(age);
12    console.log(places); //Reference error: places is not defined
13    function grandchild() {
14      // this function is linked to child() & parent() thats why name, age are accessible.
15      var places = 'Coding';
16      console.log(name);
17      console.log(age);
18      console.log(places);
19    }
20    grandchild();
21  }
22  child();
23 }
24 parent();
```

16. Explain the use of “Use Strict” in JavaScript.

Ans. “use strict” is a JavaScript directive that is introduced in Es5. The purpose of using “use strict” directive is to enforce the code is executed in strict mode. In strict mode we can’t use a variable without declaring it. “use strict” is ignored by earlier versions of JavaScript.

17. What are the different type of error present in JavaScript?

Ans. There are three types of errors available in JavaScript

Load time errors: Errors which come up when loading a web page like improper syntax errors are known as Load-time errors and it generates the errors dynamically.

Run time errors: Errors that come due to misuse of the command inside the HTML language.

Logical Errors: These are the errors that occur due to the bad logic performed on a function which is having a different operation.



18. What is Generator function in JavaScript?

Ans. A generator is a process that can be paused and resumed and can yield multiple values. A generator in JavaScript consists of a generator function, which returns an iterable Generator object.

---

JavaScript generators are just ways to make iterators. They use the yield keyword to yield execution control back to the calling function and can then resume execution once the next() function is called again

```
1  function* generator(i) {  
2      yield i;  
3      yield i + 10;  
4  }  
5  const gen = generator(10);  
6  console.log(gen.next().value);  
7  console.log(gen.next().value);
```

19. What is iife in JavaScript?

Ans. An Immediate-Invoked Function Expression (IIFE) is a function that is executed instantly after it's defined. This pattern has been used to alias global variables, make variables and functions private and to ensure asynchronous code in loops are executed correctly.

IIFE	<a href="#">Arrow function</a> IIFE	<a href="#">async</a> IIFE
<pre>(function () {     /* ... */ })();</pre>	<pre>(( ) =&gt; {     /* ... */ })();</pre>	<pre>(async () =&gt; {     /* ... */ })();</pre>



20. What is BOM in JavaScript?

Ans.

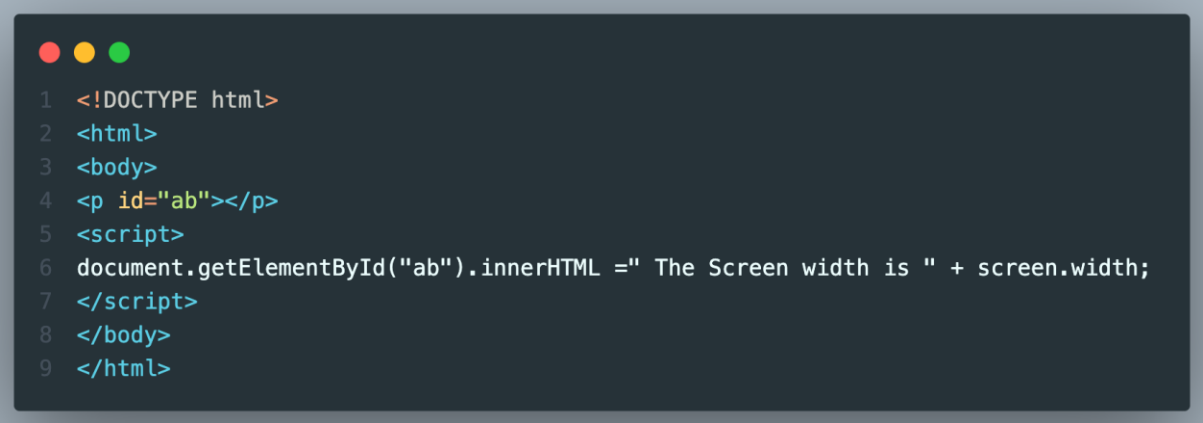
The Browser Object Model (BOM) allows JavaScript to "talk to" the browser. The BOM is a browser-specific convention referring to all the objects exposed by the web browser.

The BOM allows JavaScript to "interact with" the browser. A window object is created automatically by the browser itself.

JavaScript's window.screen object contains information about the user's screen. It can also be written without the window prefix.

Properties:

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

A screenshot of a code editor window with a dark background and light-colored text. The code is a simple HTML document with a script that updates the innerHTML of a paragraph with the screen width. The code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4 <p id="ab"></p>
5 <script>
6 document.getElementById("ab").innerHTML =" The Screen width is " + screen.width;
7 </script>
8 </body>
9 </html>
```