JAVASCRIPT Interview Questions:-

1.How do you avoid receiving postMessages from attackers

Since the listener listens for any message, an attacker can trick the application by sending a message from the attacker’s origin, which gives an impression that the receiver received the message from the actual sender’s window. You can avoid this issue by validating the origin of the message on the receiver's end using the “message.origin” attribute.

2.What are the problems with postmessage target origin as wildcard

The second argument of postMessage method specifies which origin is allowed to receive the message. If you use the wildcard “*” as an argument then any origin is allowed to receive the message. In this case, there is no way for the sender window to know if the target window is at the target origin when sending the message. If the target window has been navigated to another origin, the other origin would receive the data. Hence, this may lead to XSS vulnerabilities.

```
targetWindow.postMessage(message, "*");
```

3.Is PostMessage secure

Yes, postMessages can be considered very secure as long as the programmer/developer is careful about checking the origin and source of an arriving message. But if you try to send/receive a message without verifying its source will create cross-site scripting attacks.

4.What is the output of below spread operator array

```
[..."John Resig"];
```

The output of the array is ['J', 'o', 'h', 'n', '', 'R', 'e', 's', 'i', 'g'] Explanation: The string is an iterable type and the spread operator within an array maps every character of an

iterable to one element. Hence, each character of a string becomes an element within an Array.

5.What is for...of statement

The for…of statement creates a loop iterating over iterable objects or elements such as built-in String, Array, Array-like objects (like arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. The basic usage of for…of statement on arrays would be as below,

```
let arrayIterable = [10, 20, 30, 40, 50];

for (let value of arrayIterable) {
  value++;
  console.log(value); // 11 21 31 41 51
}
```

6.What are enhanced object literals

Object literals make it easy to quickly create objects with properties inside the curly braces. For example, it provides shorter syntax for common object property definition as below.

```
//ES6
var x = 10,
  y = 20;
obj = { x, y };
console.log(obj); // {x: 10, y:20}
```

7.How do you swap variables in destructuring assignment

If you don't use destructuring assignment, swapping two values requires a temporary variable. Whereas using a destructuring feature, two variable values can be swapped in one destructuring expression. Let's swap two number variables in array destructuring assignment,

```
var x = 10,
  y = 20;
```

```
[x, y] = [y, x];
console.log(x); // 20
console.log(y); // 10
```

8)What are default values in destructuring assignment

A variable can be assigned a default value when the value unpacked from the array or object is undefined during destructuring assignment. It helps to avoid setting default values separately for each assignment. Let's take an example for both arrays and object use cases,

Arrays destructuring:

```
var x, y, z;

[x = 2, y = 4, z = 6] = [10];
console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

Objects destructuring:

```
var { x = 2, y = 4, z = 6 } = { x: 10 };

console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

9)What is destructuring assignment

The destructuring assignment is a JavaScript expression that makes it possible to unpack values from arrays or properties from objects into distinct variables. Let's get the month values from an array using destructuring assignment

```
var [one, two, three] = ["JAN", "FEB", "MARCH"];

console.log(one); // "JAN"
console.log(two); // "FEB"
console.log(three); // "MARCH"
```

and you can get user properties of an object using destructuring assignment,

```
var { name, age } = { name: "John", age: 32 };

console.log(name); // John
console.log(age); // 32
```

10.What are raw strings

ES6 provides a raw strings feature using the `String.raw()` method which is used to get the raw string form of template strings. This feature allows you to access the raw strings as they were entered, without processing escape sequences. For example, the usage would be as below,

```
var calculationString = String.raw`The sum of numbers
is \n${
  1 + 2 + 3 + 4
}!`;
console.log(calculationString); // The sum of numbers
is \n10!
```

11.What are template literals

Template literals or template strings are string literals allowing embedded expressions. These are enclosed by the back-tick (`) character instead of double or single quotes. In ES6, this feature enables using dynamic expressions as below,

```
var greeting = `Welcome to JS World, Mr.
${firstName} ${lastName}.`;
```

11.What are default parameters

in ES6, Default function parameters feature allows parameters to be initialized with default values if no value or undefined is passed

//ES6

var calculateArea = function (height = 50, width = 60) {

  return width * height;

};

console.log(calculateArea()); //300

12. Is const variable makes the value immutable

No, the const variable doesn't make the value immutable. But it disallows subsequent assignments(i.e, You can declare with assignment but can't assign another value later)

```
const userList = [];
userList.push("John"); // Can mutate even
though it can't re-assign
```

```
console.log(userList); // ['John']
```

13.Can I redeclare let and const variables

No, you cannot redeclare let and const variables. If you do, it throws below error

```
Uncaught SyntaxError: Identifier
'someVariable' has already been declared
```

Explanation: The variable declaration with `var` keyword refers to a function scope and the variable is treated as if it were declared at the top of the enclosing scope due to hoisting feature. So all the multiple declarations contributing to the same hoisted variable without any error. Let's take an example of re-declaring variables in the same scope for both var and let/const variables.

```
var name = "John";
function myFunc() {
  var name = "Nick";
  var name = "Abraham"; // Re-assigned in the
same function block
  alert(name); // Abraham
}
myFunc();
alert(name); // John
```

The block-scoped multi-declaration throws syntax error,

```
let name = "John";
function myFunc() {
  let name = "Nick";
  let name = "Abraham"; // Uncaught
SyntaxError: Identifier 'name' has already
been declared
  alert(name);
}

myFunc();
alert(name);
```

14.What is ES6

ES6 is the sixth edition of the javascript language and it was released in June 2015. It was initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015. Almost all the modern browsers support ES6 but for the old browsers there are many transpilers, like Babel.js etc.

15.List down some of the features of ES6

Below are the list of some new features of ES6,

i. Support for constants or immutable variables
ii. Block-scope support for variables, constants and functions
iii. Arrow functions
iv. Default parameters
v. Rest and Spread Parameters
vi. Template Literals
vii. Multi-line Strings
viii. Destructuring Assignment
ix. Enhanced Object Literals
x. Promises

xi. Classes
xii. Modules

16.What is the output of below for loops

```
for (var i = 0; i < 4; i++) {
  // global scope
  setTimeout(() => console.log(i));
}

for (let i = 0; i < 4; i++) {
  // block scope
  setTimeout(() => console.log(i));
}
```

The output of the above for loops is 4 4 4 4 and 0 1 2 3

Explanation: Due to the event queue/loop of javascript, the `setTimeout` callback function is called after the loop has been executed. Since the variable i is declared with the `var` keyword it became a global variable and the value was equal to 4 using iteration when the time `setTimeout` function is invoked. Hence, the output of the first loop is `4 4 4 4`.

Whereas in the second loop, the variable i is declared as the `let` keyword it becomes a block scoped variable and it holds a new value(0, 1 ,2 3) for each iteration. Hence, the output of the first loop is `0 1 2 3`.

17.How do you create an infinite loop

You can create infinite loops using for and while loops without using any expressions. The

for loop construct or syntax is better approach in terms of ESLint and code optimizer tools,

```
for (;;) {}
while (true) {}
```

18.How to set the cursor to wait

The cursor can be set to wait in JavaScript by using the property "cursor". Let's perform this behavior on page load using the below function.

```
function myFunction() {
  window.document.body.style.cursor =
"wait";
}
```

and this function invoked on page load

```
<body onload="myFunction()"></body>
```

19.What is a void operator

The `void` operator evaluates the given expression and then returns undefined(i.e, without returning value). The syntax would be as below,

```
void expression;
void expression;
```

Let's display a message without any redirection or reload

```
<a
href="javascript:void(alert('Wel
come to JS world'))">
  Click here to see a message
</a>
```

20.Why do we call javascript as dynamic language

JavaScript is a loosely typed or a dynamic language because variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned/reassigned with values of all types.

```
let age = 50; // age is a
number now
age = "old"; // age is a
string now
age = true; // age is a
boolean
```