JAVASCRIPT INTERVIEW QUESTIONS :-

1.What is minimum timeout throttling?

In JavaScript, the minimum timeout throttling refers to the shortest duration that can be set for a setTimeout or setInterval function. The minimum practical value for the delay parameter in setTimeout and setInterval is typically around 4 milliseconds. However, the actual execution may not happen exactly at the specified delay due to browser constraints and system performance.

2.What is web speech API?

The Web Speech API is a JavaScript API that allows web developers to integrate speech recognition and synthesis capabilities into web applications.

The Web Speech API helps in creating more interactive and accessible web experiences by enabling voice-driven interactions and providing support for users who may have difficulty using traditional input methods.

3.How to cancel a fetch request?

To cancel a **`fetch`** request in JavaScript, you can use the **`AbortController`** interface, which allows you to abort or cancel the fetch operation

4.What are the different ways to deal with Asynchronous Code

Below are the list of different ways to deal with Asynchronous code.

i. Callbacks
ii. Promises
iii. Async/await
iv. Third-party libraries such as async.js,bluebird etc

5.What is AJAX

AJAX stands for Asynchronous JavaScript and XML and it is a group of related technologies(HTML, CSS, JavaScript, XMLHttpRequest API etc) used to display data asynchronously. i.e. We can send data to the server and get data from the server without reloading the web page.

6.What are wrapper objects

Primitive Values like string,number and boolean don't have properties and methods but they are temporarily converted or coerced to an object(Wrapper object) when you try to perform actions on them. For example, if you apply toUpperCase() method on a primitive string value, it does not throw an error but returns uppercase of the string.

```
let name = "john";

console.log(name.toUpperCase());
```

7.How do you disable right click in the web page?

In JavaScript, you can disable the right-click context menu on a web page by using an event listener to capture the right-click event (**`contextmenu`**) and prevent its default behavior. This approach helps prevent users from accessing the context menu via the right mouse button.

// Disable right-click context menu document.addEventListener('contextmenu', function(event) { event.preventDefault(); });

8.How do you capture browser back button?

In JavaScript, you can capture and handle the browser's back button functionality using the **`popstate`** event along with the **`window.history`** API. The **`popstate`** event is fired when the active history entry changes, typically when the user navigates through the browser history using back or forward button.

// Event listener for the popstate event window.addEventListener('popstate', function(event) { // Handle the browser back button press here console.log('Browser back button pressed'); // You can perform specific actions when the back button is pressed // For example: showing a modal, redirecting to a different page, etc. });

9.What is the easiest multi condition checking

`we can use indexOf` to compare input with multiple values instead of checking each value as one condition

```
// Array of values to compare against
const validValues = ['apple', 'banana', 'orange', 'grape'];
// Input to be checked
const userInput = 'orange';
// Check if the userInput matches any value in the array
if (validValues.indexOf(userInput) !== -1) {
 console.log('Input is valid');
} else {
  console.log('Input is not valid');
}
```

10.How do you flattening multi dimensional arrays?

Flattening bi-dimensional arrays is trivial with Spread operator.

```
const biDimensionalArr = [11, [22, 33], [44, 55], [66, 77], 88, 99];
const flattenArr = [].concat(...biDimensionalArr); // [11, 22, 33, 44, 55,
66, 77, 88, 99]
```

```
const multiDimensionalArray = [1, [2, 3], [4, [5, 6]]];

// Flatten the multi-dimensional array with a depth of 1
const flattenedArray = multiDimensionalArray.flat(1);

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

11.What is the shortcut to get timestamp?

In JavaScript, you can obtain a timestamp representing the current time in milliseconds using the **`Date`** object and its **`getTime()`** method. Alternatively, you can use **`Date.now()`** method, which is a shorter way to achieve the same result.

```
const timestamp = new Date().getTime();
console.log(timestamp); // Output: Current timestamp in milliseconds
```

const timestamp = Date.now();

console.log(timestamp); // Output: Current timestamp in milliseconds

12.How do you create copy to clipboard button

You need to select the content(using .select() method) of the input element and execute the copy command with execCommand (i.e, execCommand('copy')). You can also execute other system commands like cut and paste.

```
document.querySelector("#copy-button").onclick = function () {
  // Select the content
  document.querySelector("#copy-input").select();
  // Copy to the clipboard
  document.execCommand("copy");
};
```

13.How do you verify that an argument is a Number or not

The combination of IsNaN and isFinite methods are used to confirm whether an argument is a number or not.

```
function isNumber(n) {
  return !isNaN(parseFloat(n)) && isFinite(n);
}
```

14.How do you display data in a tabular format using console object

The `console.table()` is used to display data in the console in a tabular format to visualize complex arrays or objects.

```
const users = [
  { name: "John", id: 1, city: "Delhi" },
  { name: "Max", id: 2, city: "London" },
  { name: "Rod", id: 3, city: "Paris" },
];
console.table(users);
```

15.Is it possible to debug HTML elements in console

Yes, it is possible to get and debug HTML elements in the console just like inspecting elements.

```
const element = document.getElementsByTagName("body")[0];
console.log(element);
```

16.What is the purpose of dir method of console object

The `console.dir()` is used to display an interactive list of the properties of the specified JavaScript object as JSON.

```
const user = { name: "John", id: 1, city: "Delhi" };
console.dir(user);
```

17.What are the placeholders from console object

Below are the list of placeholders available from console object,

i. %o — It takes an object,
ii. %s — It takes a string,
iii. %d — It is used for a decimal or integer These placeholders can be represented in the console.log as below

```
const user = { name: "John", id: 1, city: "Delhi" };
console.log(
  "Hello %s, your details %o are available in the object form",
  "John",
  user
); // Hello John, your details {name: "John", id: 1, city: "Delhi"} are
available in object
```

18.How do you create an array with some data

You can create an array with some data or an array with the same values using `fill` method.

```
var newArray = new Array(5).fill("0");
console.log(newArray); // ["0", "0", "0", "0", "0"]
```

19.What is the easiest way to convert an array to an object

You can convert an array to an object with the same data using spread(...) operator.

```
var fruits = ["banana", "apple", "orange", "watermelon"];
var fruitsObject = { ...fruits };
console.log(fruitsObject); // {0: "banana", 1: "apple", 2: "orange", 3:
"watermelon"}
```

17.How do you rounding numbers to certain decimals

You can round numbers to a certain number of decimals using `toFixed` method from native javascript.

```
let pie = 3.141592653;
pie = pie.toFixed(3); // 3.142
```

18.How do you map the array values without using map method

You can map the array values without using the `map` method by just using the `from` method of Array. Let's map city names from Countries array,

```
const countries = [
  { name: "India", capital: "Delhi" },
  { name: "US", capital: "Washington" },
  { name: "Russia", capital: "Moscow" },
  { name: "Singapore", capital: "Singapore" },
  { name: "China", capital: "Beijing" },
  { name: "France", capital: "Paris" },
];

const cityNames = Array.from(countries, ({ capital }) => capital);
console.log(cityNames); // ['Delhi, 'Washington', 'Moscow', 'Singapore',
'Beijing', 'Paris']
```

19.What is destructuring aliases

Sometimes you would like to have a destructured variable with a different name than the property name. In that case, you'll use a `: newName` to specify a name for the variable. This process is called destructuring aliases.

```
const obj = { x: 1 };
// Grabs obj.x as as { otherName }
const { x: otherName } = obj;
```

20.How do you get unique values of an array

You can get unique values of an array with the combination of `Set` and rest expression/spread(...) syntax.

```
console.log([...new Set([1, 2, 4, 4, 3])]); // [1, 2, 4, 3]
```

21.How do you remove falsy values from an array

You can apply the filter method on the array by passing Boolean as a parameter. This way it removes all falsy values(0, undefined, null, false and "") from the array.

```
const myArray = [false, null, 1, 5, undefined];
myArray.filter(Boolean); // [1, 5] // is same as myArray.filter(x => x);
```

22.What is the output of prepend additive operator on falsy values

If you prepend the additive(+) operator on falsy values(null, undefined, NaN, false, ""), the falsy value converts to a number value zero. Let's display them on browser console as below,

```
console.log(+null); // 0
console.log(+undefined); // NaN
console.log(+false); // 0
console.log(+NaN); // NaN
console.log(+""); // 0
```

23.What happens if we add two arrays

If you add two arrays together, it will convert them both to strings and concatenate them. For example, the result of adding arrays would be as below,

```
console.log(["a"] + ["b"]); // "ab"
console.log([] + []); // ""
```

```
console.log(![] + []); // "false", because ![] returns false.
```

24.What happens with negating an array

Negating an array with `!` character will coerce the array into a boolean. Since Arrays are considered to be truthy So negating it will return `false`.

```
console.log(![]); // false
```

25.What is the difference between reflow and repaint

A *repaint* occurs when changes are made which affect the visibility of an element, but not its layout. Examples of this include outline, visibility, or background color. A *reflow* involves changes that affect the layout of a portion of the page (or the whole page). Resizing the browser window, changing the font, content changing (such as user typing text), using JavaScript methods involving computed styles, adding or removing elements from the DOM, and changing an element's classes are a few of the things that can trigger reflow. Reflow of an element causes the subsequent reflow of all child and ancestor elements as well as any elements following it in the DOM.

26.How to remove all line breaks from a string

The easiest approach is using regular expressions to detect and replace newlines in the string. In this case, we use replace function along with string to replace with, which in our case is an empty string.

```
function remove_linebreaks( var message ) {
    return message.replace( /[\r\n]+/gm, "" );
}
```

In the above expression, g and m are for global and multiline flags.

27.What is the output of below function calls

Code snippet:

```
const circle = {
  radius: 20,
  diameter() {
    return this.radius * 2;
  },
  perimeter: () => 2 * Math.PI * this.radius,
};
```

```
console.log(circle.diameter());
console.log(circle.perimeter());
```

Output:

The output is 40 and NaN. Remember that diameter is a regular function, whereas the value of perimeter is an arrow function. The `this` keyword of a regular function(i.e, diameter) refers to the surrounding scope which is a class(i.e, Shape object). Whereas this keyword of perimeter function refers to the surrounding scope which is a window object. Since there is no radius property on window objects it returns an undefined value and the multiple of number value returns NaN value.

28.What are asynchronous thunks?

Asynchronous thunks in JavaScript involve wrapping asynchronous actions (such as API calls or other asynchronous operations) into a function that can be dispatched within a Redux application

Asynchronous thunks in JavaScript are functions that encapsulate asynchronous operations, commonly used in middleware like Redux for managing asynchronous logic. Thunks enable delayed execution of actions, provide access to dispatch and/or state, and support complex asynchronous flows such as making API calls and handling responses. They are useful for managing side effects and dispatching actions based on the results of asynchronous tasks. Thunks provide a flexible and effective way to manage asynchronous operations in JavaScript.

29.What is a thunk function

A thunk is just a function which delays the evaluation of the value. It doesn't take any arguments but gives the value whenever you invoke the thunk. i.e, It is used not to execute now but it will be sometime in the future. Let's take a synchronous example,

```
const add = (x, y) => x + y;

const thunk = () => add(2, 3);

thunk(); // 5
```

30.Does javascript uses mixins?

No, JavaScript does not have built-in support for mixins as a language feature.

However, mixins can be implemented using various techniques, like object composition or third-party libraries, to achieve similar functionality.

31.What is the output of below console statement with unary operator

Let's take console statement with unary operator as given below,

```
console.log(+"Hello");
```

The output of the above console log statement returns NaN. Because the element is prefixed by the unary operator and the JavaScript interpreter will try to convert that element into a number type. Since the conversion fails, the value of the statement results in NaN value.

32.How do you trim a string at the beginning or ending

The `trim` method of string prototype is used to trim on both sides of a string. But if you want to trim especially at the beginning or ending of the string then you can use `trimStart/trimLeft` and `trimEnd/trimRight` methods. Let's see an example of these methods on a greeting message,

```
var greeting = "   Hello, Goodmorning!   ";

console.log(greeting); // "   Hello, Goodmorning!   "
console.log(greeting.trimStart()); // "Hello, Goodmorning!   "
console.log(greeting.trimLeft()); // "Hello, Goodmorning!   "

console.log(greeting.trimEnd()); // "   Hello, Goodmorning!"
console.log(greeting.trimRight()); // "   Hello, Goodmorning!"
```

33.How do you return all matching strings against a regular expression

The `matchAll()` method can be used to return an iterator of all results matching a string against a regular expression. For example, the below example returns an array of matching string results against a regular expression,

```
let regexp = /Hello(\d?))/g;
let greeting = "Hello1Hello2Hello3";

let greetingList = [...greeting.matchAll(regexp)];

console.log(greetingList[0]); //Hello1
console.log(greetingList[1]); //Hello2
console.log(greetingList[2]); //Hello3
```

34.How do you create specific number of copies of a string

The `repeat()` method is used to construct and return a new string which contains the specified number of copies of the string on which it was called, concatenated together. Remember that this method has been added to the ECMAScript 2015 specification. Let's take an example of Hello string to repeat it 4 times,

```
"Hello".repeat(4); // 'HelloHelloHelloHello'
```

35.What is the difference between Shallow and Deep copy

There are two ways to copy an object,

Shallow Copy: Shallow copy is a bitwise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

Example

```
var empDetails = {
  name: "John",
  age: 25,
  expertise: "Software Developer",
};
```

to create a duplicate

```
var empDetailsShallowCopy = empDetails; //Shallow copying!
```

if we change some property value in the duplicate one like this:

```
empDetailsShallowCopy.name = "Johnson";
```

The above statement will also change the name of `empDetails`, since we have a shallow copy. That means we're losing the original data as well.

Deep copy: A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

Example

```
var empDetails = {
  name: "John",
```

```
  age: 25,
  expertise: "Software Developer",
};
```

Create a deep copy by using the properties from the original object into new variable

```
var empDetailsDeepCopy = {
  name: empDetails.name,
  age: empDetails.age,
  expertise: empDetails.expertise,
};
```

Now if you change `empDetailsDeepCopy.name`, it will only affect `empDetailsDeepCopy` & not `empDetails`

36.How do you combine two or more arrays

The concat() method is used to join two or more arrays by returning a new array containing all the elements. The syntax would be as below,

```
array1.concat(array2, array3, ..., arrayX)
```

Let's take an example of array's concatenation with veggies and fruits arrays,

```
var veggies = ["Tomato", "Carrot", "Cabbage"];
var fruits = ["Apple", "Orange", "Pears"];
var veggiesAndFruits = veggies.concat(fruits);
console.log(veggiesAndFruits); // Tomato, Carrot, Cabbage, Apple, Orange,
Pears
```

37.What is the purpose of some method in arrays

The some() method is used to test whether at least one element in the array passes the test implemented by the provided function. The method returns a boolean value. Let's take an example to test for any odd elements,

```
var array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var odd = (element) => element % 2 !== 0;

console.log(array.some(odd)); // true (the odd element exists)
```

38.What is the difference between a parameter and an argument

Parameter is the variable name of a function definition whereas an argument represents the value given to a function when it is invoked.

function greet(name) { // Here, `name` is a parameter

  console.log(`Hello, ${name}!`);

}

```
greet('John'); // 'John' is an argument passed to the `greet` function
```

39.Do all objects have prototypes

No. All objects have prototypes except for the base object which is created by the user, or an object that is created using the new keyword.

40.What are the list of cases error thrown from non-strict mode to strict mode

Assignment to Undeclared Variables: In strict mode, assigning values to variables without declaring them with **`var`**, **`let`**, or **`const`** will throw a **`ReferenceError`**

```
'use strict';
undeclaredVar = 10; // Throws ReferenceError: undeclaredVar is not defined
```

Eval and Arguments as Variables: In strict mode, **`eval`** and **`arguments`** cannot be used as variable names or for assignments. Attempting to assign values to them will throw a **`SyntaxError`**.

- 'use strict';
  var eval = 10; // Throws SyntaxError: Unexpected eval or arguments in strict mode
- `When you declare a function in a block`
- **`When you use delete operator on a variable name`**

41.What is the output of below string expression

```
console.log("Welcome to JS world"[0]);
```

The output of the above expression is "W". Explanation: The bracket notation with specific index on a string returns the character at a specific location. Hence, it returns the character "W" of the string. Since this is not supported in IE7 and below versions, you may need to use the .charAt() method to get the desired result.

42.What is ArrayBuffer

An ArrayBuffer object is used to represent a generic, fixed-length raw binary data buffer. You can create it as below,

```
let buffer = new ArrayBuffer(16); // create a buffer of length 16
alert(buffer.byteLength); // 16
```

To manipulate an ArrayBuffer, we need to use a “view” object.

```
//Create a DataView referring to the buffer
let view = new DataView(buffer);
```

43.How do you convert character to ASCII code

You can use the `String.prototype.charCodeAt()` method to convert string characters to ASCII numbers. For example, let's find ASCII code for the first letter of 'ABC' string,

```
"ABC".charCodeAt(0); // returns 65
```

Whereas `String.fromCharCode()` method converts numbers to equal ASCII characters.

```
String.fromCharCode(65, 66, 67); // returns 'ABC'
```

44.What is the purpose of double tilde operator

The double tilde operator(~~) is known as double NOT bitwise operator. This operator is a slightly quicker substitute for Math.floor().

let num = 3.75;

let integer = ~~num; // Double tilde operator used for converting to integer

console.log(integer); // Output: 3

45.How do you get the status of a checkbox

You can apply the `checked` property on the selected checkbox in the DOM. If the value is `true` it means the checkbox is checked, otherwise it is unchecked. For example, the below HTML checkbox element can be access using javascript as below:

```
<input type="checkbox" id="checkboxname" value="Agree" /> Agree
the
conditions<br />
```

```
console.log(document.getElementById('checkboxname').checked);
// true or false
```

46.Is JavaScript faster than server side script

Yes, JavaScript is faster than server side scripts. Because JavaScript is a client-side script it does not require any web server's help for its computation or calculation. So JavaScript is always faster than any server-side script like ASP, PHP, etc.

47.What is the difference between internal and external javascript

Internal JavaScript: It is the source code within the script tag. External JavaScript: The source code is stored in an external file(stored with .js extension) and referred with in the tag.

48.What paradigm is Javascript

JavaScript is a multi-paradigm language, supporting imperative/procedural programming, Object-Oriented Programming and functional programming. JavaScript supports Object-Oriented Programming with prototypical inheritance.

49.Is postMessages synchronous

No, **`postMessage`** in JavaScript is an asynchronous method used for communication between different windows or between a window and its embedded iframe.

50.Can I avoid using postMessages completely?

You cannot avoid using postMessages completely(or 100%). Even though your application doesn't use postMessage considering the risks, a lot of third party scripts use postMessage to

communicate with the third party service. So your application might be using postMessage without your knowledge.

However, it's important to note that if your application involves communication between different origins, **`postMessage`** is the recommended and secure way to facilitate cross-origin communication while adhering to browser security policies.