JavaScript Interview Questions and Answers

1. What is Javascript?

JavaScript is a scripting language most often used for client-side and server-side web development.

2. Explain Hoisting in javascript

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

Example:

```
console.log(message); // Output: undefined
var message = 'Hello, hoisting!';

sayHello(); // Output: Hello, hoisting!

function sayHello() {
  console.log('Hello, hoisting!');
}
```

3. What is the difference between == and ===?

"==" checks equality only,

"===" checks for equality as well as the type.

4. What is NaN property in JavaScript?

NaN property represents the “Not-a-Number” value. It indicates a value that is not a legal number. Type of NaN will return a Number.

To check if a value is NaN, we use the is NaN() function,

Example:

```
isNaN("Hello")  // Returns true
isNaN(345)   // Returns false
```

5. What is DOM?

DOM stands for the Document Object Model. Dom defines a standard for accessing the document. It is a platform that allows to dynamically access and update the content or structure or style of the document.

6. What is the difference between undefined value and null value?

- **Undefined value:** A value that is not defined and has no keyword is known as undefined value.

  Example:

  ```
  int number;//Here, a number has an undefined value.
  ```

- **Null value:** A value that is explicitly specified by the keyword "null" is known as a null value.

  Example:

  ```
  String str=null;//Here, str has a null value.
  ```

7. What would be the result of 1+2+'3'

The result is 33

8. How do you change the style of a HTML element?

```
document. getElementById("myText").style.fontSize = "10";
```

9. What is Prompt() method in JavaScript?

A prompt box is a box which allows the user to enter input by providing a text box. The prompt() method displays a dialog box that prompts the visitor for input.

10. What is typeof operator?

The typeof operator is a built-in JavaScript operator that allows you to determine the data type of a given value or expression. It returns a string indicating the type of the operand.

Example:

```
let x = 42;
let y = "Hello";
let z = true;

console.log(typeof x);  // Output: "number"
console.log(typeof y);  // Output: "string"
console.log(typeof z);  // Output: "boolean"
```

11. What is let keyword in JavaScript?

The let keyword is used to declare block-scoped variables. It was introduced in ECMAScript 6 (ES6) as an alternative to the var keyword, which declares variables with function scope or global scope.

The let keyword allows you to declare variables that are limited in scope to the block, statement, or expression in which they are defined. This means that a variable declared with let is only accessible within the block of code where it is defined, including any nested blocks. Outside of that block, the variable is not accessible.

Example:

```
function example() {
  let x = 10;

  if (true) {
    let y = 20;
    console.log(x);  // Output: 10
    console.log(y);  // Output: 20
  }

  console.log(x);    // Output: 10
  console.log(y);    // Error: y is not defined
}

example();
```

12. What is var keyword in JavaScript?

The var keyword is used to declare variables. It was the primary way to declare variables in JavaScript before the introduction of let and const in ECMAScript 6 (ES6).

Variables declared with var have function scope or global scope, depending on where they are declared. This means that a variable declared with var is accessible throughout the entire function in which it is defined, regardless of the block scope.

Example:

```
function example() {
  var x = 10;

  if (true) {
    var y = 20;
    console.log(x);  // Output: 10
    console.log(y);  // Output: 20
  }

  console.log(x);    // Output: 10
  console.log(y);    // Output: 20
}

example();
```

13. What is const keyword in JavaScript?

The const keyword is used to declare variables that have block scope and whose values cannot be reassigned once they are initialized. It was introduced in ECMAScript 6 (ES6) as a way to declare constants in JavaScript.

When you declare a variable with const, you must initialize it with a value at the same time. Once the value is assigned, it cannot be changed throughout the rest of the program.

Example:

```
const pi = 3.14;
```

```
console.log(pi);   // Output: 3.14

pi = 3.14159;      // Error: Assignment to a constant variable
```

14. What is arrays in JavaScript?

In JavaScript, an array is a data structure used to store multiple values in a single variable. It is a built-in object type that provides a way to organize and manipulate collections of elements.

There are two ways to create array in JavaScript like other languages.

- The first way to create array

```
var names = new Array ();
names [0] = "Vikas";
names [1] = "Ashish";
names [2] = "Nikhil";
```

- The second way to create array

```
var names = new Array ("Vikas", "Ashish", "Nikhil");
```

15. What arrow functions?

Arrow functions, also known as fat arrow functions, are a concise syntax introduced in ECMAScript 6 (ES6) for defining functions in JavaScript. They provide a more concise and expressive way to write function expressions.

Syntax:

```
(parameters) => {
  // function body
};
```

Example:

```
const add = (a, b) => a + b;
```

```
console.log(add(2, 3));   // Output: 5
```

16. What are object? Explain How to add and remove elements from object.

An object is a data structure that allows you to store and organize data in key-value pairs.

An object consists of properties, where each property has a key (also called a property name or identifier) and a corresponding value. The key is always a string, and the value can be of any data type, including other objects, arrays, functions, and primitive values such as strings, numbers, and booleans.

Example:

```
let person = {
  name: "John",
  age: 30,
  profession: "Developer"
};
delete person.age;
```

17. How to add JavaScript to HTML document? Explain each way with examples.

There are three ways to add JavaScript code to an HTML document.

- **Inline javascript**

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline JavaScript Example</title>
```

```
</head>
<body>
  <h1>Inline JavaScript Example</h1>

  <script>
    // Inline JavaScript code
    alert("Hello, World!");
  </script>
</body>
</html>
```

- **Internal javascript**

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JavaScript Example</title>
  <script>
    // Internal JavaScript code
    function greet() {
      alert("Hello, World!");
    }
  </script>
</head>
<body>
  <h1>Internal JavaScript Example</h1>

  <button onclick="greet()">Click Me</button>
</body>
</html>
```

- **External javascript**

```
<!DOCTYPE html>
<html>
<head>
  <title>External JavaScript Example</title>
  <script src="script.js"></script>
</head>
<body>
  <h1>External JavaScript Example</h1>

  <button onclick="greet()">Click Me</button>
</body>
</html>
```

script.js

```
// External JavaScript code
function greet() {
  alert("Hello, World!");
}
```

18. Who created javascript?

Javascript is created by Brendan Eich. He developed the language while working at Netscape Communications Corporation in 1995.

19. What are the different ways to access object properties?

In JavaScript, there are several ways to access object properties.

- **Dot notation:**

```
let person = {
  name: "John",
  age: 30
};

console.log(person.name);  // Output: "John"
console.log(person.age);   // Output: 30
```

- **Bracket notation:**

```
let person = {
  name: "John",
  age: 30
};

console.log(person["name"]);  // Output: "John"
console.log(person["age"]);   // Output: 30
```

- **Computed property access (ES6):**

```
let propertyName = "name";
let person = {
  name: "John",
  age: 30
};

console.log(person[propertyName]);  // Output: "John"
```

- **Object destructuring (ES6):**

```
let person = {
  name: "John",
  age: 30
};

let { name, age } = person;

console.log(name);  // Output: "John"
console.log(age);   // Output: 30
```

20. Is JavaScript a case-sensitive language

Yes, JavaScript is a case-sensitive language.

Example:

```
let myVariable = 42;
let myvariable = "Hello";

console.log(myVariable);   // Output: 42
console.log(myvariable);   // Output: "Hello"
```

21. What advantages are using arrow functions?

- Arrow functions have shorter syntax than regular function expressions.
- Arrow functions have implicit return statements.
- Arrow functions increase readability.

22. What are the primitive data types in JavaScript?

In JavaScript, there are six primitive data types:

- **String:** Represents a sequence of characters enclosed in single quotes (") or double quotes ("").
  Example: "Hello, world!"

- **Number:** Represents numeric values, including integers and floating-point numbers.
  Example: 42, 3.14

- **Boolean:** Represents a logical value that can be either true or false.
  Example: true, false

- **Null:** Represents the intentional absence of any object value.
  Example: null

- **Undefined:** Represents an uninitialized or unassigned value.
  Example: undefined

- Symbol (introduced in ECMAScript 6): Represents a unique and immutable value that can be used as an identifier for object properties.
  Example: Symbol("mySymbol")

23. What are some advantages of using External JavaScript?

- It separates HTML and code.
- It makes HTML and JavaScript easier to read and maintain.
- Cached JavaScript files can speed up page loads.

24. Is javascript a statically typed or a dynamically typed language?

JavaScript is a dynamically typed language. This means that variable types are determined dynamically at runtime, rather than being explicitly declared or enforced during compilation or initialization.

In JavaScript, you can assign a value of any type to a variable without explicitly specifying its type. The type of a variable can change during the execution of the program based on the value assigned to it.

Example:

```
let x = 42;   // x is initially assigned a number
x = "Hello";  // x is now assigned a string
x = true;     // x is now assigned a Boolean
```

25. What are the possible ways to create objects in JavaScript?

In JavaScript, there are four ways to create an object — using object literals, constructor functions, ES6 classes and object.

- **Object Literal:**

  Example:

  ```
  let person = {
    name: "John",
    age: 25,
    occupation: "Developer"
  };
  ```

- **Constructor Function:**

  Example:

  ```
  function Person(name, age, occupation) {
    this.name = name;
    this.age = age;
  ```

```
  this.occupation = occupation;
}

let person = new Person("John", 25, "Developer");
```

- **ES6 Class:**

  Example:

  ```
  class Person {
    constructor(name, age, occupation) {
      this.name = name;
      this.age = age;
      this.occupation = occupation;
    }
  }

  let person = new Person("John", 25, "Developer");
  ```

- **Object.create():**

  Example:

  ```
  let personPrototype = {
    greeting: function() {
      console.log("Hello, I'm " + this.name);
    }
  };

  let person = Object.create(personPrototype);
  person.name = "John";
  person.age = 25;
  person.occupation = "Developer";
  ```

26. What is JavaScript? Explain the features of JavaScript.

JavaScript is a scripting language most often used for client-side and server-side web development.

The features of JavaScript are:

- **Case Sensitive**

  In Javascript, names, variables, keywords, and functions are case-sensitive.

- **Arrow Functions**

  Javascript helps to optimize syntax in anonymous functions with the arrow function syntax.

- **Date and Time Handling**

  Javascript has built-in functions for getting 'date' and time.

- **Event Handling**

  Events are actions. Javascript provides event-handling options.

- **Control Statements**

  Javascript has control statements like if-else-if, switch case, and loop. Users can write complex code using these control statements.

- **Scripting**

  Javascript executes the client-side script in the browser.

27. What are events?

Event is the predefined object in js where it contains all the information related to click event.

28. What is the purpose of the array slice method?

The slice() method in JavaScript is used to create a new array that contains a shallow copy of a portion of an existing array. It does not modify the original array but returns a new array with the selected elements.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const slicedArray = numbers.slice(1, 4);

console.log(slicedArray); // Output: [2, 3, 4]
```

29. What is the purpose of the array splice method?

The splice() method in JavaScript is used to change the contents of an array by removing, replacing, or adding elements. It modifies the original array and returns an array containing the removed elements.

Example:

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

// Remove one element at index 1
fruits.splice(1, 1);
console.log(fruits); // Output: ['apple', 'orange', 'grape']
```

30. What are classes in javascript?

Classes in JavaScript are a way to define blueprints for creating objects with similar properties and behaviors. They are a fundamental part of object-oriented programming (OOP) in JavaScript. Introduced in ECMAScript 2015 (ES6)

Example:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

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

31. What is the definition of a Higher-Order Function?

A higher order function is a function that takes one or more functions as arguments, or returns a function as its result.

Example:

```
function multiplier(factor) {
  return function(number) {
    return number * factor;
  };
}

const double = multiplier(2);
const triple = multiplier(3);

console.log(double(5)); // Output: 10
console.log(triple(5)); // Output: 15
```

32. What are the different types of operators in JavaScript? Explain each type with an Example.

In JavaScript, there are several types of operators that perform different operations on values. Here are the main types of operators along with examples:

- **Arithmetic Operators:** Arithmetic operators are used to perform mathematical calculations.

  Example:

  ```
  let x = 5;
  let y = 3;

  console.log(x + y); // Addition: 8
  console.log(x - y); // Subtraction: 2
  console.log(x * y); // Multiplication: 15
  console.log(x / y); // Division: 1.6666666666666667
  console.log(x % y); // Modulus (Remainder): 2
  console.log(x ** y); // Exponentiation: 125
  ```

- **Comparison Operators:** Comparison operators are used to compare two values and return a Boolean result (true or false).

  Example:

  ```
  let x = 5;
  let y = 3;

  console.log(x > y);  // Greater than: true
  console.log(x < y);  // Less than: false
  console.log(x >= y); // Greater than or equal to: true
  ```

```
console.log(x <= y); // Less than or equal to: false
console.log(x === y); // Equality: false
console.log(x !== y); // Inequality: true
```

- **Logical Operators:** Logical operators are used to combine or manipulate Boolean values.
  Example:

```
let x = 5;
let y = 3;
let z = 7;

console.log(x > y && x < z); // Logical AND: true
console.log(x > y || x > z); // Logical OR: true
console.log(!(x > y));       // Logical NOT: false
```

- **Unary Operators:** Unary operators work on a single operand.
  Example:

```
let x = 5;

console.log(-x);   // Negation: -5
console.log(++x);  // Increment: 6
console.log(--x);  // Decrement: 5
console.log(!true); // Logical NOT: false
```

- **Conditional (Ternary) Operator:** The conditional operator (also known as the ternary operator) is a shorthand for an if...else statement.
  Example:

```
let age = 18;
let canVote = (age >= 18) ? "Yes" : "No";
```

```
console.log(canVote); // Yes
```

- **Assignment Operators:** Assignment operators are used to assign values to variables.
  Example:

```
let x = 10;

x += 5; // Addition assignment: x = x + 5 (15)
x -= 3; // Subtraction assignment: x = x - 3 (12)
x *= 2; // Multiplication assignment: x = x * 2 (24)
x /= 4; // Division assignment: x = x / 4 (6)
x %= 5; // Modulus assignment: x = x % 5 (1)
```

33. Explain is Scope in JavaScript?

Scope in JavaScript refers to the visibility and accessibility of variables, functions, and objects within a particular part of the code during runtime. It determines where variables and functions are accessible and where they are not.

- **Global Scope:** Variables declared outside of any function or block have global scope. They can be accessed from anywhere in the code, including inside functions.
  Example:

```
let globalVariable = "I am a global variable";

function globalFunction() {
  console.log(globalVariable);
}
```

```
globalFunction(); // Output: I am a global variable
```

- **Local Scope:** Variables declared inside a function have local scope. They are only accessible within that function and are not visible outside of it.
  Example:

```
function localFunction() {
  let localVariable = "I am a local variable";
  console.log(localVariable);
}

localFunction(); // Output: I am a local variable
console.log(localVariable); // Error: localVariable is not defined
```

34. What are the types of errors in javascript?

In JavaScript, there are several types of errors that can occur during the execution of a program.

- **Syntax Errors:** Syntax errors occur when the JavaScript code violates the language's syntax rules.

  Example:

```
if (x > 5 {  // SyntaxError: Missing closing parenthesis
  console.log("x is greater than 5");
}
```

- **Reference Errors:** Reference errors occur when an invalid reference or identifier is used in the code.

  Example:

```
console.log(message); // ReferenceError: message is not defined
```

```
function myFunction() {
  console.log(innerVariable); // ReferenceError: innerVariable is not defined
}
```

- **Type Errors:** Type errors occur when an operation is performed on a value of an inappropriate type.

  Example:

  ```
  let x = 10;
  x(); // TypeError: x is not a function

  let person = null;
  console.log(person.name); // TypeError: Cannot read property 'name' of null
  ```

- **Range Errors:** Range errors occur when a value is not within the expected range or set of allowed values.

  Example:

  ```
  let array = [1, 2, 3];
  console.log(array[5]); // RangeError: Invalid array index

  let negativeNumber = -10;
  console.log(Math.sqrt(negativeNumber)); // RangeError: Invalid argument
  ```

- **Eval Errors:** Eval errors occur when there is an issue with the eval() function. The eval() function is used to evaluate JavaScript code dynamically.

  Example:

```
eval("alert('Hello, World!"); // EvalError: Unterminated string literal
```

35. Is JavaScript a compiled or interpreted language

JavaScript is an interpreted language, not a compiled language

36. What is the use of setInterval?

The setInterval() function is commonly used to set a delay for functions that are executed again and again, such as animations.

37. What is the purpose of clearTimeout method?

The clearTimeout method in JavaScript is used to cancel a timeout previously set with the setTimeout function. It allows you to stop the execution of a function that was scheduled to run after a specific delay.

Example:

```
function showMessage() {
  console.log("Hello, World!");
}

// Schedule the showMessage function to execute after 2 seconds
let timeoutId = setTimeout(showMessage, 2000);

// Cancel the timeout before it executes
clearTimeout(timeoutId);
```

38. What is the purpose of clearInterval method?

The clearInterval method in JavaScript is used to cancel the recurring execution of a function that was scheduled to run at fixed intervals using the setInterval function.

Example:

```
let count = 0;

function incrementCount() {
  count++;
  console.log(count);
}

// Call incrementCount function every 1 second (1000 milliseconds)
let intervalId = setInterval(incrementCount, 1000);

// Stop the interval after 5 seconds
setTimeout(function() {
  clearInterval(intervalId);
}, 5000);
```

39. What is the difference between slice and splice?

slice	splice
Returns removed elements from the array	Returns selected elements from the array
Mutates original array	Does not mutate original array
Can add new elements to array	Can't add new elements

40. What is eval?

In JavaScript, eval() is a global function that evaluates or executes a string of code dynamically at runtime. It takes the provided code as a string parameter and executes it as if it were part of the original code.

Example:

```
let x = 5;
let y = 10;
let code = "console.log(x + y);";

eval(code); // Output: 15

let dynamicCode = "let z = x * y; console.log(z);";

eval(dynamicCode); // Output: 50
```

41. What is the purpose of setTimeout function?

The setTimeout function in JavaScript is used to schedule the execution of a function or the evaluation of a code snippet after a specified delay. It allows you to introduce a time delay before executing a specific action.

Example:

```
function showMessage() {
  console.log("Hello, World!");
}

// Call showMessage function after 2 seconds
setTimeout(showMessage, 2000);

// Using an anonymous function
setTimeout(function() {
  console.log("Delayed message");
}, 3000);
```

42. How do you get the current url with javascript?

In JavaScript, you can get the current URL (Uniform Resource Locator) of the webpage using the window.location object. The window.location object provides information about the current URL and various properties and methods to access different parts of the URL.

Example:

```
let currentURL = window.location.href;
console.log(currentURL);
```

43. How do you combine two or more arrays?

To combine two or more arrays in JavaScript, you can use the concat() method or the spread operator (...).

Example:

```
let array1 = [1, 2, 3];
let array2 = [4, 5, 6];
let array3 = [7, 8, 9];

let combinedArray = array1.concat(array2, array3);
console.log(combinedArray);
// Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

44. Can I redeclare let and const variables

In JavaScript, you cannot redeclare a variable using the let or const keywords within the same scope. If you attempt to redeclare a let or const variable, it will result in an error.

Example:

```
let x = 5;
let x = 10; // Error: Identifier 'x' has already been declared

const y = 20;
const y = 30; // Error: Identifier 'y' has already been declared
```

45. What is an anonymous function?

An anonymous function in JavaScript is a function that is defined without a name. Instead of providing a name for the function, it is directly declared as an expression and assigned to a variable or used as a callback.

Example:

```
let sayHello = function() {
  console.log("Hello!");
};

sayHello(); // Output: Hello!
```

46. How do you display the current date in javascript?

To display the current date in JavaScript, you can use the Date object along with various methods to extract the desired information. Here's an

Example:

```
let currentDate = new Date();

let year = currentDate.getFullYear();
let month = currentDate.getMonth() + 1; // Note: Months are zero-based,
so we add 1 to get the actual month.
let day = currentDate.getDate();

console.log("Current Date: " + year + "-" + month + "-" + day);
```

47. Explain the different Output statements in JavaScript with Examples.

In JavaScript, there are several ways to output or display information to the console or the user.

- **console.log():**

  ```
  console.log("Hello, World!"); // Output: Hello, World!

  let num = 42;
  console.log("The value of num is:", num); // Output: The value of num is: 42

  let person = { name: "John", age: 30 };
  console.log(person); // Output: { name: "John", age: 30 }
  ```

- **alert():**

  ```
  alert("Welcome to our website!"); // Displays a popup with the message "Welcome to our website!"
  ```

- **document.write():**

  ```
  document.write("Hello, World!"); // Output: Hello, World!
  ```

- **innerHTML:**

  ```
  <div id="myElement"></div>

  <script>
    let element = document.getElementById("myElement");
    element.innerHTML = "Hello, World!"; // The content of the div will be replaced with "Hello, World!"
  ```

```
</script>
```

48. What are Events in JS? Briefly Explain Event Handlers in JS with Examples.

In JavaScript, events are actions or occurrences that happen in the browser or on a web page. These events can be triggered by the user, such as clicking a button or submitting a form, or they can be triggered by the browser itself, such as when the page finishes loading or when a timer expires. Events allow you to respond to user interactions and perform specific actions in your JavaScript code.

Example:

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

<script>
  // Event handler function
  function handleClick() {
    console.log("Button clicked!");
  }

  // Attaching the event handler to the button
  let button = document.getElementById("myButton");
  button.addEventListener("click", handleClick);
</script>
```

49. What are functions? Explain the different types of function with example

In JavaScript, a function is a block of reusable code that performs a specific task or calculates a value.

- **Named Function:**

A named function is a function that is defined with a specific name. It can be called by its name whenever you want to execute its code.

Example:

```
function greet(name) {
  console.log("Hello, " + name + "!");
}

greet("John"); // Output: Hello, John!
```

- **Anonymous Function:**

  An anonymous function is a function that is not assigned a name. It is often used as a callback function or assigned to a variable.

  Example:

```
let greet = function (name) {
  console.log("Hello, " + name + "!");
};

greet("John"); // Output: Hello, John!
```

- **Arrow Function:**

  Arrow functions are a concise syntax introduced in ES6. They provide a more compact way to define functions

  Example:

```
let greet = (name) => {
  console.log("Hello, " + name + "!");
};
```

```
greet("John"); // Output: Hello, John!
```

50. Explain with example About looping and control statements in JavaScript.

In JavaScript, looping and control statements are used to control the flow of execution in a program. They allow you to repeat a block of code or conditionally execute code based on certain conditions.

- **for loop:**

  ```
  for (let i = 0; i < 5; i++) {
    console.log(i);
  }
  // Output: 0 1 2 3 4
  ```

- **while loop:**

  ```
  let i = 0;
  while (i < 5) {
    console.log(i);
    i++;
  }
  // Output: 0 1 2 3 4
  ```

- **do-while loop:**

  ```
  let i = 0;
  do {
    console.log(i);
    i++;
  } while (i < 5);
  ```

```
// Output: 0 1 2 3 4
```

- **if statement:**

```
let age = 20;
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
// Output: You are an adult.
```

- **switch statement:**

```
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("It's Monday.");
    break;
  case "Tuesday":
    console.log("It's Tuesday.");
    break;
  default:
    console.log("It's another day.");
}
// Output: It's Monday.
```

51. What is DOM in JS? Explain Each DOM methods with Examples.

In JavaScript, the DOM (Document Object Model) is a programming interface that represents the structure and content of an HTML or XML document. It provides methods and properties that allow you to manipulate and interact with the elements on a web page.

- **getElementById():**

The getElementById() method is used to select an element from the DOM based on its unique ID attribute.

Example:

```
<div id="myDiv">Hello, World!</div>

<script>
  let element = document.getElementById("myDiv");
  console.log(element.innerHTML); // Output: Hello, World!
</script>
```

- **getElementsByClassName():**

The getElementsByClassName() method is used to select elements from the DOM based on their class names. It returns a collection of elements.

Example:

```
<p class="highlight">This is a paragraph.</p>
<p class="highlight">This is another paragraph.</p>

<script>
  let elements = document.getElementsByClassName("highlight");
  for (let i = 0; i < elements.length; i++) {
    console.log(elements[i].innerHTML);
  }
  // Output:
  // This is a paragraph.
  // This is another paragraph.
</script>
```

- **getElementsByTagName():**

The getElementsByTagName() method is used to select elements from the DOM based on their tag names. It returns a collection of elements.

Example:

```
<p>This is a paragraph.</p>
<div>This is a div.</div>
<p>This is another paragraph.</p>

<script>
  let elements = document.getElementsByTagName("p");
  for (let i = 0; i < elements.length; i++) {
    console.log(elements[i].innerHTML);
  }
  // Output:
  // This is a paragraph.
  // This is another paragraph.
</script>
```

- **querySelector():**

The querySelector() method is used to select an element from the DOM using a CSS selector. It returns the first matching element.

Example:

```
<div class="myDiv">Hello, World!</div>

<script>
  let element = document.querySelector(".myDiv");
  console.log(element.innerHTML); // Output: Hello, World!
</script>
```

52. What are callbacks?

In JavaScript, a callback is a function that is passed as an argument to another function and is invoked or called within that function.

53. How do you submit a form using JavaScript?

To submit a form using JavaScript, you can utilize the submit() method of the HTMLFormElement object.

Example:

```
<form id="myForm">
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <input type="submit" value="Submit">
</form>

<script>
  // Get the form element
  const form = document.getElementById('myForm');

  // Add an event listener to the form's submit event
  form.addEventListener('submit', function(event) {
    event.preventDefault(); // Prevent the default form submission behavior

    // Perform any necessary form validation or other actions

    // Submit the form programmatically
    form.submit();
  });
</script>
```

54. How do you check whether a string contains a substring?

In JavaScript, there are multiple ways to check whether a string contains a substring. Here are a few common approaches:

- **Using the includes() method:**

The includes() method is a built-in method for strings that returns true if the specified substring is found within the string, and false otherwise.

Example:

```
const str = 'Hello, world!';
const substring = 'world';

console.log(str.includes(substring)); // Output: true
```

- **Using the indexOf() method:**

The indexOf() method returns the index of the first occurrence of a substring within a string. If the substring is not found, it returns -1. You can use this method to check if the returned index is greater than or equal to 0.

Example:

```
const str = 'Hello, world!';
const substring = 'world';

console.log(str.indexOf(substring) >= 0); // Output: true
```

55. How do you validate an email in javascript?

To validate an email address in JavaScript, you can use regular expressions. Regular expressions provide a powerful way to match and validate patterns.

Example:

```
function validateEmail(email) {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
}

// Example usage
const email1 = 'test@example.com';
const email2 = 'invalid.email@com';
const email3 = 'another@example';

console.log(validateEmail(email1)); // Output: true
console.log(validateEmail(email2)); // Output: false
console.log(validateEmail(email3)); // Output: false
```

56. Give an example where do you really need semicolon

In JavaScript, semicolons are used to terminate statements. While JavaScript has automatic semicolon insertion (ASI) rules that insert semicolons in certain cases, there are scenarios where using semicolons explicitly is necessary.

Example:

```
// Example 1: Function as an Immediately Invoked Function Expression (IIFE)
(function() {
  console.log('This is an IIFE');
})();

// Example 2: Multiple statements on the same line
const name = 'John Doe'; console.log('Hello, ' + name);

// Example 3: Using semicolons to separate statements
const x = 5;
const y = 10;
const z = x + y;

console.log(z);
```

57. How do you extend classes?

In JavaScript, you can extend classes using the extends keyword to create a subclass or derived class. The derived class inherits properties and methods from the parent class.

Example:

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a sound.`);
  }
}
```

```
class Dog extends Animal {
  constructor(name, breed) {
    super(name); // Call the parent class constructor using super()
    this.breed = breed;
  }

  speak() {
    console.log(`${this.name} barks!`);
  }
}

// Create instances of the classes
const animal = new Animal('Animal');
animal.speak(); // Output: "Animal makes a sound."

const dog = new Dog('Buddy', 'Labrador');
dog.speak(); // Output: "Buddy barks!"
```

58. Explain arrays in JavaScript. Explain methods of Array with example

Arrays in JavaScript are used to store multiple values in a single variable. They are ordered, indexed collections of values that can be of any data type, such as numbers, strings, objects, or even other arrays. Arrays in JavaScript are dynamic, meaning their size can change dynamically by adding or removing elements.

- **push():** Adds one or more elements to the end of an array and returns the new length of the array.

  Example:

  ```
  const fruits = ['apple', 'banana'];
  fruits.push('orange');
  console.log(fruits); // Output: ['apple', 'banana', 'orange']
  ```

- **pop():** Removes the last element from an array and returns that element.

  Example:

  ```
  const fruits = ['apple', 'banana', 'orange'];
  const removedFruit = fruits.pop();
  console.log(removedFruit); // Output: 'orange'
  console.log(fruits); // Output: ['apple', 'banana']
  ```

- **shift():** Removes the first element from an array and returns that element.

  Example:

  ```
  const fruits = ['apple', 'banana', 'orange'];
  const removedFruit = fruits.shift();
  console.log(removedFruit); // Output: 'apple'
  console.log(fruits); // Output: ['banana', 'orange']
  ```

- **unshift():** Adds one or more elements to the beginning of an array and returns the new length of the array.

  Example:

  ```
  const fruits = ['banana', 'orange'];
  fruits.unshift('apple');
  console.log(fruits); // Output: ['apple', 'banana', 'orange']
  ```

- **slice():** Returns a new array containing a portion of the original array, specified by start and end indices.
  Example:

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

  const slicedFruits = fruits.slice(1, 4);

  console.log(slicedFruits); // Output: ['banana', 'orange', 'grape']
  ```

- **concat():** Joins two or more arrays and returns a new array.
  Example:

  ```
  const fruits1 = ['apple', 'banana'];
  const fruits2 = ['orange', 'grape'];
  const combinedFruits = fruits1.concat(fruits2);
  console.log(combinedFruits); // Output: ['apple', 'banana', 'orange',
  'grape']
  ```

59. Explain Strings in JavaScript. Explain methods of Strings with example

Strings in JavaScript are sequences of characters enclosed in single quotes ('') or double quotes (""). They are used to represent text and can be manipulated using various string methods provided by the String object.

- **length:** Returns the length of a string.

Example:

```
const str = 'Hello, World!';

console.log(str.length); // Output: 13
```

- **charAt():** Returns the character at a specified index in a string.

Example:

```
const str = 'Hello, World!';
console.log(str.charAt(7)); // Output: 'W'
```

- **substring():** Returns a portion of a string based on start and end indices

Example:

```
const str = 'Hello, World!';
console.log(str.substring(7, 12)); // Output: 'World'
```

- **toUpperCase():** Converts a string to uppercase.

Example:

```
const str = 'Hello, World!';

console.log(str.toUpperCase()); // Output: 'HELLO, WORLD!'
```

- **toLowerCase():** Converts a string to lowercase.

Example:

```
const str = 'Hello, World!';

console.log(str.toLowerCase()); // Output: 'hello, world!'
```

- **concat():** Concatenates two or more strings.

Example:

```
const str1 = 'Hello,';

const str2 = ' World!';

console.log(str1.concat(str2)); // Output: 'Hello, World!'
```

- **indexOf():** Returns the index of the first occurrence of a specified substring within a string.

Example:

```
const str = 'Hello, World!';

console.log(str.indexOf('World')); // Output: 7
```

- **replace():** Replaces a specified value or substring with another value.

Example:

const str = 'Hello, World!';

console.log(str.replace('World', 'JavaScript')); // Output: 'Hello, JavaScript!'

- **split():** Splits a string into an array of substrings based on a specified separator.

Example:

const str = 'Hello, World!';

console.log(str.split(', ')); // Output: ['Hello', 'World!']

60. How do you redirect new page in javascript?

To redirect to a new page in JavaScript, you can use the window.location object's href property or the assign() method.

- **Using window.location.href:**
  window.location.href = 'https://www.example.com';

- **Using window.location.assign():**
  window.location.assign('https://www.example.com');

- **Using window.location.replace():**
  window.location.replace('https://www.example.com');

61. How do you check if a key exists in an object?

To check if a key exists in an object in JavaScript, you can use the hasOwnProperty() method or the in operator.

- **Using the hasOwnProperty() method:**

  const obj = { name: 'John', age: 25 };

  console.log(obj.hasOwnProperty('name')); // Output: true

  console.log(obj.hasOwnProperty('gender')); // Output: false

- **Using the in operator:**

  ```
  const obj = { name: 'John', age: 25 };

  console.log('name' in obj); // Output: true
  console.log('gender' in obj); // Output: false
  ```

62. What is an arguments object?

In JavaScript, the arguments object is a built-in object available within the scope of a function. It contains an array-like collection of arguments passed to the function when it is invoked.

Example:

```
function sum() {
  let total = 0;
  for (let i = 0; i < arguments.length; i++) {
    total += arguments[i];
  }
  return total;
}


console.log(sum(1, 2, 3)); // Output: 6
console.log(sum(4, 5, 6, 7)); // Output: 22
```

63. How do you test for an empty object?

To test if an object is empty in JavaScript, you can check if it has any own properties using the Object.keys() method or by checking the object's length.

Example:

```
const obj = {};


if (Object.keys(obj).length === 0) {
```

```
  console.log("The object is empty");
} else {
  console.log("The object is not empty");
}
```

64. What is this keyword in javascript?

In JavaScript, the this keyword refers to the object that is currently executing the code or the context in which the code is being executed. It is a special identifier that allows you to access properties and methods within the scope of that object.

Example:

```
const obj = {
  name: "John",
  sayHello: function() {
    console.log(`Hello, ${this.name}!`);
  }
};

obj.sayHello(); // Output: Hello, John!
```

65. what are callback fuctions in javascript?

In JavaScript, a callback function is a function that is passed as an argument to another function and is executed at a later point in time or in response to an event. Callback functions are a way to ensure that certain code is executed only after a particular task or operation is completed.

Example:

```
function doMath(a, b, callback) {
  const sum = a + b;
  const difference = a - b;
  // Call the callback function with the calculated values
  callback(sum, difference);
}

// Define a callback function
function handleResults(sum, difference) {
  console.log("Sum:", sum);
  console.log("Difference:", difference);
}

// Call the doMath function with the handleResults callback
doMath(10, 5, handleResults);
```

66. Why JavaScript is dynamically typed language? Explain the uses of JavaScript.

JavaScript is considered a dynamically typed language because it allows variables to hold values of any data type, and the type of a variable can be changed during runtime. This means that you don't have to explicitly declare the type of a variable before using it, and the type of a variable can change based on the value assigned to it.

- **Web development:** JavaScript is primarily used to make web pages interactive, handle user interactions, and create dynamic content on websites.

- **Client-side scripting:** JavaScript runs directly in the web browser, allowing you to validate forms, manipulate HTML elements, and enhance the user experience.

- **Front-end frameworks:** JavaScript frameworks like React, Angular, and Vue.js enable the development of complex and interactive user interfaces for web applications.

- **Server-side development:** With Node.js, JavaScript can be used on the server-side to build scalable and high-performance applications, handling server logic and database operations.

- **Game development**: JavaScript, along with libraries and game engines, can be used to create browser-based games and even mobile games.

67. What is the use of a constructor function in javascript?

In JavaScript, a constructor function is used to create and initialize objects of a particular class or type. It serves as a blueprint for creating multiple instances of objects with similar properties and methods.

Example:

```
function Person(name, age) {
  this.name = name;
  this.age = age;

  this.greet = function() {
    console.log('Hello, my name is ' + this.name + ' and I am ' + this.age + '
years old.');
  };
}
```

```
// Creating instances of Person using the constructor function
var person1 = new Person('John', 30);
var person2 = new Person('Jane', 25);

// Accessing properties and invoking methods
console.log(person1.name); // Output: John
console.log(person2.age); // Output: 25
person1.greet(); // Output: Hello, my name is John and I am 30 years old.
person2.greet(); // Output: Hello, my name is Jane and I am 25 years old.
```

68. Is there any relation between Java and JavaScript

JavaScript has no direct relation to Java besides being used for web technologies. The name choice was a marketing move to encourage adoption.

69. How do you trim a string in javascript?

In JavaScript, you can trim a string by removing any leading and trailing whitespace characters (such as spaces, tabs, or line breaks) from it. JavaScript provides two methods for trimming strings: trim() and trimStart()/trimEnd().

- **trim():** The trim() method removes whitespace from both the beginning and the end of a string.
- **Example:**

```
const str = '   Hello, World!   ';
const trimmedStr = str.trim();
console.log(trimmedStr); // Output: 'Hello, World!'
```

- **trimStart() and trimEnd():** These methods allow you to trim whitespace from either the start or end of a string, respectively. They are available starting from ECMAScript 2021 (ES12).

  Example:

  ```
  const str = '   Hello, World!   ';
  const trimmedStartStr = str.trimStart();
  const trimmedEndStr = str.trimEnd();
  console.log(trimmedStartStr); // Output: 'Hello, World!   '
  console.log(trimmedEndStr); // Output: '   Hello, World!'
  ```