**JAVASCRIPT BASIC&DOM**

**1.What is JavaScript?**

Ans. JavaScript is a versatile programming language commonly used for web development to add interactivity and dynamic content to websites. It runs on the client side, enabling the creation of interactive user interfaces and enhancing the overall user experience.

**2.what is the use of isNaN function?**

Ans. The `isNaN` function in JavaScript is used to check if a value is NaN (Not a Number). It returns `true` if the provided value is NaN, and `false` if the value is a valid number or can be converted to one. It's often employed to validate user input or to handle situations where a value might be expected to be a number.

**3.What is negative infinity?**

Ans. Negative infinity, denoted as `-Infinity`, is a mathematical concept and a special value in JavaScript. It represents a quantity that is infinitely small, extending indefinitely in the negative direction on the number line. In programming, it is often encountered in situations where a calculation results in a value that is too small to be represented by the data type used, or in specific mathematical operations that approach negative infinity as a limit.

**4.Which company developed JavaScript?**

Ans. JavaScript was developed by Netscape Communications Corporation. It was created by Brendan Eich in 1995 while he was working at Netscape. Initially, it was named "Mocha" and later "LiveScript" before being officially named JavaScript.

**5.What are undeclared and undefined variables?**

Ans. An undeclared variable is a variable that has not been declared using a `var`, `let`, or `const` keyword before it is used. Attempting to use an undeclared variable can lead to errors in JavaScript.

An undefined variable, on the other hand, is a variable that has been declared but has not been assigned a value. When you try to access the value of an undefined variable, it will result in the special value `undefined`.

Example:

```javascript

// Undeclared variable

Console.log(undeclaredVariable); // Results in an error

// Declared but undefined variable

Let undefinedVariable;

Console.log(undefinedVariable); // Outputs: undefined

```

It’s good practice to always declare variables before using them to avoid potential issues and to initialize variables with a value if you intend to use them later in your code.

**6.Write the code for adding new elements dynamically?**

Ans. Certainly! You can dynamically add new elements to the HTML document using JavaScript. Here’s a simple example using the DOM (Document Object Model):

```html

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Dynamically Add Elements</title>

</head>

<body>

<!—Existing HTML content 🡪

<div id=”container”>

<p>Existing content</p>

</div>

<!—JavaScript to dynamically add elements 🡪

<script>

// Create a new element

Var newElement = document.createElement(“p”);

// Set content for the new element

newElement.textContent = “Dynamically added content!”;

// Get reference to existing container

Var container = document.getElementById(“container”);

// Append the new element to the container

Container.appendChild(newElement);

</script>

</body>

</html>

```

This example creates a new paragraph (`<p>`) element, sets its content, and then appends it to an existing `div` with the id “container”. You can modify this code to suit your specific needs, creating different types of elements and adjusting their properties as necessary.

**7.What is the difference between ViewState and SessionState?**

Ans. In the context of web development, ViewState and SessionState are terms commonly associated with ASP.NET, a web application framework. Here's the difference between them:

1. \*\*ViewState:\*\*

- \*\*Scope:\*\* ViewState is used to persist the state of a single page across postbacks. It's maintained on the client side, typically in a hidden field within the HTML.

- \*\*Data Storage:\*\* ViewState stores data for a specific page and is sent back and forth between the server and the client with each request and response.

- \*\*Lifetime:\*\* It has a short lifespan, tied to the duration of a user's interaction with a particular page.

2. \*\*SessionState:\*\*

- \*\*Scope:\*\* SessionState, on the other hand, is used to persist data across multiple pages during a user's session.

- \*\*Data Storage:\*\* SessionState is stored on the server side, usually in memory or an external storage mechanism, and is associated with a unique user session identified by a session ID.

- \*\*Lifetime:\*\* It persists for a longer duration than ViewState, typically for the entire duration of a user's interaction with the web application until the session expires.

In summary, ViewState is focused on maintaining the state of a single page across postbacks, while SessionState is designed for persisting data across multiple pages and throughout a user's session.

**8. What is === operator?**

Ans. The `===` operator in JavaScript is called the strict equality operator. It compares two values for equality without performing type coercion. This means that not only do the values need to be equal, but they also must be of the same data type.

For example:

```javascript

3 === 3 // true

‘3’ === 3 // false (string is not equal to number)

```

In contrast, the loose equality operator `==` performs type coercion, attempting to convert operands to the same type before making the comparison. Using `===` is generally considered good practice as it avoids unexpected type conversions and can help prevent subtle bugs in your code.

**9. How can the style/class of an element be changed?**

Ans.You can change the style or class of an HTML element using JavaScript. Here are examples for both scenarios:

1. \*\*Changing Style:\*\*

- You can directly manipulate the `style` property of an element. Here’s an example that changes the color of a paragraph with the id “myParagraph” to red:

```javascript

Document.getElementById(“myParagraph”).style.color = “red”;

```

2. \*\*Changing Class:\*\*

- If you want to change the class of an element, you can use the `classList` property. Assuming you have a div with the id “myDiv” and you want to change its class to “newClass”:

```javascript

Document.getElementById(“myDiv”).classList.add(“newClass”);

```

Or, if you want to remove a class:

```javascript

Document.getElementById(“myDiv”).classList.remove(“oldClass”);

```

Remember to replace “myParagraph”, “myDiv”, “red”, “newClass”, and “oldClass” with the actual IDs, colors, or class names you are working with in your HTML document.

**10. How to read and write a file using JavaScript?**

Ans. In a web browser environment, JavaScript doesn’t have direct access to the file system for security reasons. However, you can read and write files using JavaScript in other environments such as Node.js or by utilizing browser APIs like the File API for reading files selected by the user.

Here's an example of how to read a file selected by the user in a web browser:

```html

<input type=”file” id=”fileInput”>

<script>

Document.getElementById(‘fileInput’).addEventListener(‘change’, function(event) {

Const file = event.target.files[0];

Const reader = new FileReader();

Reader.onload = function(event) {

Const contents = event.target.result;

Console.log(‘File contents:’, contents);

// You can perform further processing with the file contents here

};

Reader.readAsText(file);

});

</script>

```

And for writing files, if you’re working in a Node.js environment, you can use the built-in `fs` module:

```javascript

Const fs = require(‘fs’);

// Write content to a file

Fs.writeFile(‘example.txt’, ‘Hello, world!’, function (err) {

If (err) throw err;

Console.log(‘File saved!’);

});

// Read content from a file

Fs.readFile(‘example.txt’, ‘utf8’, function(err, data) {

If (err) throw err;

Console.log(‘File contents:’, data);

});

```

Remember that in a web browser environment, file read and write operations are generally limited to files selected by the user and require user interaction.

**11.What are all the looping structures in JavaScript?**

Ans. JavaScript supports several looping structures for executing a block of code repeatedly. Here are the main looping constructs:

1. \*\*for loop:\*\*

```javascript

For (initialization; condition; iteration) {

// code to be repeated

}

```

1. \*\*while loop:\*\*

```javascript

While (condition) {

// code to be repeated

}

```

1. \*\*do-while loop:\*\*

```javascript

Do {

// code to be repeated

} while (condition);

```

1. \*\*for…in loop:\*\*

Used for iterating over the enumerable properties of an object.

```javascript

For (variable in object) {

// code to be repeated

}

```

1. \*\*for…of loop:\*\*

Introduced in ECMAScript 2015, used for iterating over iterable objects (arrays, strings, etc.).

```javascript

For (variable of iterable) {

// code to be repeated

}

```

Each type of loop has its own use cases, and you can choose the one that best fits the requirements of your code.

**12.How can you convert the string of any base to an integer in JavaScript?**

Ans. In JavaScript, you can use the `parseInt` function to convert a string representing a number in a specific base to an integer. The second argument of `parseInt` specifies the base of the numeral system.

Here's an example of converting a binary string to an integer:

```javascript

const binaryString = "101010";

const decimalNumber = parseInt(binaryString, 2);

console.log(decimalNumber); // Outputs: 42

```

In this example, `parseInt` is used with a base of `2` to indicate that the input string is in binary. Similarly, you can use other bases like `8` for octal or `16` for hexadecimal.

```javascript

const octalString = "52";

const decimalNumberOctal = parseInt(octalString, 8);

console.log(decimalNumberOctal); // Outputs: 42

```

```javascript

const hexString = "2A";

const decimalNumberHex = parseInt(hexString, 16);

console.log(decimalNumberHex); // Outputs: 42

```

Make sure to specify the correct base when using `parseInt` to ensure accurate conversion.

**13. What is the function of the delete operator?**

Ans. The `delete` operator in JavaScript is used to delete properties from objects or elements from an array. Its functionality depends on the context in which it is used:

1. \*\*Deleting Object Properties:\*\*

```javascript

Const myObject = { key1: ‘value1’, key2: ‘value2’ };

Delete myObject.key1;

```

This will remove the `key1` property from the `myObject` object.

1. \*\*Deleting Array Elements:\*\*

```javascript

Const myArray = [1, 2, 3, 4, 5];

Delete myArray[2];

```

This will remove the element at index `2` from the `myArray` array. However, it leaves an undefined slot in the array, and the array length doesn’t change.

It's Important to note that the `delete` operator is not intended for use with variables or functions, and it doesn’t free up memory. It is primarily used for manipulating properties of objects and elements of arrays.

**14.What are all the types of Pop up boxes available in JavaScript?**

Ans.In JavaScript, there are three main types of pop-up boxes that you can use to interact with users:

1. \*\*Alert Box:\*\*

- Displays a message with an “OK” button.

```javascript

Alert(“This is an alert message!”);

```

2. \*\*Confirm Box:\*\*

- Prompts the user with a message and provides “OK” and “Cancel” buttons. It returns `true` if the user clicks “OK” and `false` if the user clicks “Cancel.”

```javascript

Const userConfirmed = confirm(“Are you sure?”);

```

3. \*\*Prompt Box:\*\*

- Similar to the confirm box but includes an input field where the user can enter some text. It returns the entered text if the user clicks “OK” and `null` if the user clicks “Cancel.”

```javascript

Const userInput = prompt(“Please enter something:”, “Default text”);

```

These pop-up boxes can be useful for obtaining user input or displaying important messages, but their usage should be done judiciously to ensure a good user experience.

**15.What is the use of Void (0)?**

Ans. The use of `void(0)` in JavaScript is often seen in the context of anchor (`<a>`) tags with the `href` attribute. It is used to create a link that doesn’t navigate to a new page or reload the current page when clicked.

For example:

```html

<a href=”javascript:void(0);” onclick=”myFunction()”>Click me</a>

```

In this case, clicking the link triggers the `myFunction()` JavaScript function without causing the browser to change the page. The `void(0)` is used to evaluate an expression that always returns `undefined`, essentially preventing any page navigation.

It's worth noting that In modern web development, it’s generally recommended to use alternative methods, such as event listeners or preventing the default behavior, rather than relying on `javascript:void(0)`. For example:

```html

<a href=”#” onclick=”myFunction(); return false;”>Click me</a>

```

Or, using an event listener:

```html

<a href=”#” id=”myLink”>Click me</a>

<script>

Document.getElementById(“myLink”).addEventListener(“click”, function(event) {

Event.preventDefault();

myFunction();

});

</script>

```

These approaches achieve the same goal without relying on the `void(0)` construct.

**16.How can a page be forced to load another page in JavaScript?**

Ans. In JavaScript, you can force a page to load another page by manipulating the `window.location` object. There are a couple of ways to achieve this:

1. \*\*Using `window.location.href`:\*\*

```javascript

// Redirect to another page

Window.location.href = <https://www.example.com>;

```

1. \*\*Using `window.location.replace()`:\*\*

```javascript

// Redirect to another page, replacing the current one in the browser’s history

Window.location.replace(<https://www.example.com>);

```

In both cases, when the JavaScript code is executed, the browser will navigate to the specified URL. Keep in mind that navigating away from the current page might disrupt the user experience, so it’s crucial to use this approach judiciously and provide appropriate feedback to users.

Additionally, it’s common to trigger such redirects in response to user actions or specific conditions in your code.

**17.What are the disadvantages of using innerHTML in JavaScript?**

Ans. While the `innerHTML` property in JavaScript is convenient for manipulating HTML content, it does have some disadvantages:

1. \*\*Security Concerns:\*\*

- Using `innerHTML` can expose your application to security risks like Cross-Site Scripting (XSS) if you insert user-generated content without proper validation or sanitization. Malicious scripts may be injected and executed.

2. \*\*Performance Overhead:\*\*

- Manipulating `innerHTML` tends to be slower than other methods like DOM manipulation using createElement, appendChild, etc. This is because when you set `innerHTML`, the browser has to parse and render the entire HTML content, potentially impacting performance.

3. \*\*Event Listeners and Data Binding:\*\*

- If you replace or modify content using `innerHTML`, you might lose event listeners or data bindings associated with the elements being replaced. This can lead to unexpected behavior or the need to rebind events after each `innerHTML` modification.

4. \*\*Limited to HTML Markup:\*\*

- `innerHTML` is specifically designed for HTML content. If you're working with XML or need to manipulate non-HTML content, using other DOM manipulation methods might be more appropriate.

5. \*\*Readability and Maintenance:\*\*

- Overuse of `innerHTML` can make your code less readable and maintainable, especially in larger applications. Using more structured DOM manipulation methods can make your code more predictable and easier to maintain.

In summary, while `innerHTML` is a powerful and convenient tool, it's essential to be aware of its limitations and potential risks. In many cases, using other DOM manipulation methods may be more appropriate, depending on the specific requirements of your application.

**18.Write a program to Show an alert**.

Ans.

<script>

// JavaScript code to show an alert

Alert(“Hello, this is an alert!”);

</script>

**19.What will be the result for these expressions?**

1. 5 > 4

2. "apple" > "pineapple"

3. "2" > "12"

4. undefined == null

5. undefined === null

6. null == "\n0\n"

7. 7. null === +"\n0\n"

Ans. Here are the results for the given expressions:

1. `5 > 4`: True

2. `"apple" > "pineapple"`: False (Lexicographically, "apple" comes before "pineapple")

3. `"2" > "12"`: True (Lexicographically, "2" comes after "1")

4. `undefined == null`: True

5. `undefined === null`: False (Strict equality checks both value and type)

6. `null == "\n0\n"`: False (Loose equality, null is not equal to any string)

7. `null === +"\n0\n"`: False (Strict equality, different types)

**19.Will alert be shown?**

If (“0”) { alert( ‘Hello’); }

Ans. Yes, the alert will be shown. In JavaScript, the condition `”0”` evaluates to true in a boolean context. Therefore, the code inside the `if` statement will be executed, and the alert will be displayed.

20. What is the code below going to output? alert( null || 2 || undefined );

Ans. The code will output `2`. In a logical OR (`||`) chain, the expression returns the first truthy value encountered or the last value if all values are falsy. In this case, `null` is falsy, but `2` is truthy, so the expression evaluates to `2`, and the alert will display `2`.

20. The following function returns true if the parameter age is greater than

18. Otherwise it asks for a confirmation and returns its result:

function

checkAge(age)

{

else {

}

}

if (age> 18) { return true; }

// ...return confirm (‘did parents allow you?');

Ans. It looks like there’s a syntax issue in the provided code. Here’s a corrected version:

```javascript

Function checkAge(age) {

If (age > 18) {

Return true;

} else {

Return confirm(‘Did parents allow you?’);

}

}

```

This function checks if the `age` is greater than 18. If true, it returns `true`. Otherwise, it asks for confirmation and returns the result of the confirmation dialog.

**21.Replace Function Expressions with arrow functions in the code below:**

Function

Ask(question, yes, no)

{ if (confirm(question))yes();

Else

No();

}

Ask(“Do you agree?”, function()

{ alert(“You agreed.”); },

Function() {

Alert(“You canceled the execution.”); }

}

Ans. const ask = (question, yes, no) => {

if (confirm(question)) yes();

else no();

};

ask(

"Do you agree?",

() => alert("You agreed."),

() => alert("You canceled the execution.")

);

**22.Write the code, one line for each action:**

1. **Create an empty object user.**
2. **Add the property name with the value John.**
3. **Add the property surname with the value Smith.**
4. **Change the value of the name to Pete.**
5. **Remove the property name from the object.**

Ans. // a) Create an empty object user.

Let user = {};

// b) Add the property name with the value John.

User.name = “John”;

// c) Add the property surname with the value Smith.

User.surname = “Smith”;

// d) Change the value of the name to Pete.

User.name = “Pete”;

// e) Remove the property name from the object.

Delete user.name;

**23.Is array copied?**

Let fruits = [“Apples”, “Pear”, “Orange”]; // push a new value into the “copy” let

shoppingCart = fruits; shoppingCart.push(“Banana”); // what’s in fruits?

Alert( fruits.length ); // ?

Ans. No, the array is not copied. Both `fruits` and `shoppingCart` reference the same array, so pushing a value into one will affect the other. After executing the provided code, `fruits` will contain four elements: [“Apples”, “Pear”, “Orange”, “Banana”], and `alert(fruits.length)` will output `4`.

**24.Map to names**

Let john = { name: “John”, age: 25 }; let pete = { name: “Pete”, age: 30 }; let mary =

{ name: “Mary”, age: 28 }; let users = [ john, pete, mary ]; let names = /\* … your

code \*/ alert( names ); // John, Pete, Mary

Ans. let john = { name: "John", age: 25 };

let pete = { name: "Pete", age: 30 };

let mary = { name: "Mary", age: 28 };

let users = [john, pete, mary];

let names = users.map(user => user.name);

alert(names); // John, Pete, Mary

25. Map to objects

let john = { name: "John", surname: "Smith", id: 1 }; let pete = { name: "Pete",

surname: "Hunt", id: 2 }; let mary = { name: "Mary", surname: "Key", id: 3 }; let

users = [ john, pete, mary ]; let usersMapped = /\* ... your code ... \*/

/\*

usersMapped = [

{ fullName: “John Smith”, id: 1 },

{ fullName: “Pete Hunt”, id: 2 },

{ fullName: “Mary Key”, id: 3 }

]

\*/ alert( usersMapped[0].id ) // 1 alert( usersMapped[0].fullName ) // John Smith

Ans. let john = { name: "John", surname: "Smith", id: 1 };

let pete = { name: "Pete", surname: "Hunt", id: 2 };

let mary = { name: "Mary", surname: "Key", id: 3 };

let users = [john, pete, mary];

let usersMapped = users.map(user => ({

fullName: `${user.name} ${user.surname}`,

id: user.id

}));

alert(usersMapped[0].id); // 1

alert(usersMapped[0].fullName); // John Smith

**26.Sum the properties There is a salaries object with arbitrary number of salaries. Write The function sumSalaries(salaries) that returns the sum of all salaries using Object.values and the for..of loop.If salaries is empty, then the result must be 0.**

**Let salaries = {**

**“John”: 100,**

**“Pete”: 300,**

**“Mary”: 250**

**};**

**Alert( sumSalaries(salaries) ); // 650**

Ans. Function sumSalaries(salaries) {

Let sum = 0;

For (let salary of Object.values(salaries)) {

Sum += salary;

}

Return sum;

}

Let salaries = {

“John”: 100,

“Pete”: 300,

“Mary”: 250

};

Alert(sumSalaries(salaries)); // 650

**27.Destructuring assignment We have an object: Write the Destructuring assignment that**

**Reads:**

1. **Name property into the variable name.**
2. **Year’s property into the variable age.**
3. **isAdmin property into the variable isAdmin (false, if no such property)**
4. **let user = { name: “John”, years: 30};**

Ans. Let user = { name: “John”, years: 30 };

// a) Name property into the variable name.

Let { name } = user;

// b) Year’s property into the variable age.

Let { years: age } = user;

// c) isAdmin property into the variable isAdmin (false, if no such property).

Let { isAdmin = false } = user;

**28.Turn the object into JSON and back Turn the user into JSON and then read it back**

**Into another variable.**

**User = { name: “John Smith”, age: 35};**

Ans.let user = { name: “John Smith”, age: 35 };

// Turn the object into JSON

Let userJSON = JSON.stringify(user);

// Read it back into another variable

Let anotherUser = JSON.parse(userJSON);

// Now ‘anotherUser’ contains the data from the original ‘user’ object

**29.Create a program to hide/show the password.**

Ans. Certainly! Here’s a simple example of a program using HTML and JavaScript to toggle the visibility of a password input field:

```html

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Password Toggle</title>

<style>

Body {

Font-family: Arial, sans-serif;

}

</style>

</head>

<body>

<label for=”password”>Password:</label>

<input type=”password” id=”password” placeholder=”Enter your password”>

<button id=”toggleButton”>Show/Hide Password</button>

<script>

Const passwordInput = document.getElementById(“password”);

Const toggleButton = document.getElementById(“toggleButton”);

toggleButton.addEventListener(“click”, function() {

// Toggle the type attribute between “password” and “text”

passwordInput.type = passwordInput.type === “password” ? “text” : “password”;

});

</script>

</body>

</html>

```

This program includes an input field for the password, a button to toggle visibility, and JavaScript to handle the toggle functionality. When you click the “Show/Hide Password” button, it changes the input type between “password” and “text,” making the password either visible or hidden.

**30. Create a program that will select all the classes and loop over and whenever i click the button the alert should show.**

Ans. <!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Class Selector</title>

<style>

.sample-class {

Padding: 10px;

Margin: 5px;

Border: 1px solid #ccc;

Cursor: pointer;

}

</style>

</head>

<body>

<div class=”sample-class”>Element 1</div>

<div class=”sample-class”>Element 2</div>

<div class=”sample-class”>Element 3</div>

<button id=”showAlertButton”>Click Me</button>

<script>

// Select all elements with the class “sample-class”

Const elements = document.querySelectorAll(‘.sample-class’);

// Add click event listener to each element

Elements.forEach(element => {

Element.addEventListener(‘click’, function() {

Alert(`Clicked on: ${this.textContent}`);

});

});

// Add click event listener to the button

Const showAlertButton = document.getElementById(‘showAlertButton’);

showAlertButton.addEventListener(‘click’, function() {

alert(‘Button Clicked!’);

});

</script>

</body>

</html>

31. Create a responsive header using proper JavaScript.

Ans. <!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Responsive Header</title>

<style>

body {

margin: 0;

font-family: Arial, sans-serif;

}

header {

background-color: #333;

color: #fff;

padding: 10px;

text-align: center;

}

nav {

display: flex;

justify-content: space-around;

background-color: #444;

padding: 10px;

display: none;

}

nav a {

color: #fff;

text-decoration: none;

}

@media screen and (max-width: 600px) {

nav {

flex-direction: column;

align-items: center;

}

}

</style>

</head>

<body>

<header>

<h1>Responsive Header</h1>

<button id="mobileMenuButton">Toggle Menu</button>

</header>

<nav>

<a href="#">Home</a>

<a href="#">About</a>

<a href="#">Contact</a>

</nav>

<script>

document.getElementById('mobileMenuButton').addEventListener('click', function() {

const nav = document.querySelector('nav');

nav.style.display = nav.style.display === 'flex' ? 'none' : 'flex';

});

</script>

</body>

</html>

**32. Create a form and validate using JavaScript.**

Ans. <!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Form Validation</title>

<style>

Body {

Font-family: Arial, sans-serif;

Padding: 20px;

}

Label {

Display: block;

Margin-bottom: 5px;

}

Input {

Margin-bottom: 10px;

Padding: 5px;

Width: 100%;

}

Button {

Padding: 10px;

Background-color: #4CAF50;

Color: white;

Border: none;

Cursor: pointer;

}

</style>

</head>

<body>

<form id=”myForm”>

<label for=”username”>Username:</label>

<input type=”text” id=”username” name=”username” required>

<label for=”email”>Email:</label>

<input type=”email” id=”email” name=”email” required>

<label for=”password”>Password:</label>

<input type=”password” id=”password” name=”password” required>

<button type=”button” onclick=”validateForm()”>Submit</button>

</form>

<script>

Function validateForm() {

Const username = document.getElementById(‘username’).value;

Const email = document.getElementById(‘email’).value;

Const password = document.getElementById(‘password’).value;

// Simple validation, you can add more conditions as needed

If (username === ‘’ || email === ‘’ || password === ‘’) {

Alert(‘Please fill in all fields.’);

} else {

Alert(‘Form submitted successfully!’);

// You can also submit the form to a server or perform other actions here.

}

}

</script>

</body>

</html>

**33. Create a modal box using css and Js with three buttons**

Ans. <!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Modal Box</title>

<style>

Body {

Font-family: Arial, sans-serif;

Display: flex;

Align-items: center;

Justify-content: center;

Height: 100vh;

Margin: 0;

}

#overlay {

Display: none;

Position: fixed;

Top: 0;

Left: 0;

Width: 100%;

Height: 100%;

Background: rgba(0, 0, 0, 0.5);

Align-items: center;

Justify-content: center;

}

#modal {

Background-color: #fff;

Padding: 20px;

Border-radius: 5px;

Box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);

}

Button {

Padding: 10px;

Margin: 5px;

Cursor: pointer;

}

</style>

</head>

<body>

<button onclick=”openModal()”>Open Modal</button>

<div id=”overlay”>

<div id=”modal”>

<p>This is a modal box.</p>

<button onclick=”closeModal()”>Close</button>

<button onclick=”buttonClicked(1)”>Button 1</button>

<button onclick=”buttonClicked(2)”>Button 2</button>

<button onclick=”buttonClicked(3)”>Button 3</button>

</div>

</div>

<script>

Function openModal() {

Document.getElementById(‘overlay’).style.display = ‘flex’;

}

Function closeModal() {

Document.getElementById(‘overlay’).style.display = ‘none’;

}

Function buttonClicked(buttonNumber) {

Alert(`Button ${buttonNumber} clicked!`);

}

</script>

</body>

</html>

**34.Use external js library to show slider.**

Ans. Certainly! One popular external JavaScript library for creating sliders is `Slick Carousel`. Below is an example of how you can use the `Slick Carousel` library to create a simple slider:

1. Include the `Slick Carousel` CSS and JavaScript files in the `<head>` section of your HTML:

```html

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Slider with Slick Carousel</title>

<link rel=”stylesheet” type=”text/css” href=<https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick.min.css>/>

<link rel=”stylesheet” type=”text/css” href=<https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick-theme.min.css>/>

</head>

<body>

```

1. Add your HTML structure for the slider:

```html

<div class=”slider”>

<div><img src=”image1.jpg” alt=”Slide 1”></div>

<div><img src=”image2.jpg” alt=”Slide 2”></div>

<div><img src=”image3.jpg” alt=”Slide 3”></div>

</div>

```

1. Include the `Slick Carousel` JavaScript file and initialize the slider in an external script file:

```html

<script src=<https://code.jquery.com/jquery-3.6.0.min.js>></script>

<script src=<https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick.min.js>></script>

<script src=”yourscript.js”></script>

</body>

</html>

```

1. In your external JavaScript file (`yourscript.js`), initialize the slider:

```javascript

$(document).ready(function(){

$(‘.slider’).slick({

Dots: true,

Infinite: true,

Speed: 500,

slidesToShow: 1,

slidesToScroll: 1

});

});

```

Ensure that you replace `”image1.jpg”`, `”image2.jpg”`, and `”image3.jpg”` with the actual paths to your images.

**35. Prevent the browser when i click the form submit button**

Ans. <!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Prevent Form Submission</title>

<style>

Body {

Font-family: Arial, sans-serif;

Padding: 20px;

}

Label {

Display: block;

Margin-bottom: 5px;

}

Input {

Margin-bottom: 10px;

Padding: 5px;

Width: 100%;

}

Button {

Padding: 10px;

Background-color: #4CAF50;

Color: white;

Border: none;

Cursor: pointer;

}

</style>

</head>

<body>

<form id=”myForm”>

<label for=”username”>Username:</label>

<input type=”text” id=”username” name=”username” required>

<label for=”email”>Email:</label>

<input type=”email” id=”email” name=”email” required>

<label for=”password”>Password:</label>

<input type=”password” id=”password” name=”password” required>

<button type=”button” onclick=”validateForm()”>Submit</button>

</form>

<script>

Function validateForm() {

Const username = document.getElementById(‘username’).value;

Const email = document.getElementById(‘email’).value;

Const password = document.getElementById(‘password’).value;

// Simple validation, you can add more conditions as needed

If (username === ‘’ || email === ‘’ || password === ‘’) {

Alert(‘Please fill in all fields.’);

} else {

Alert(‘Form submitted successfully!’);

// You can also submit the form to a server or perform other actions here.

}

}

Document.getElementById(‘myForm’).addEventListener(‘submit’, function(event) {

Event.preventDefault(); // Prevent the default form submission

validateForm(); // Call your validation function or perform other actions here

});

</script>

</body>

</html>

**36. What is JSON?**

Ans. JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript programming language and is commonly used for transmitting data between a server and a web application as an alternative to XML.

JSON data consists of key-value pairs, where keys are strings and values can be strings, numbers, arrays, objects, booleans, or null. It is often used to represent structured data such as objects and arrays.

JSON has become a standard format for data interchange due to its simplicity, versatility, and wide support across programming languages and platforms. It is commonly used in web development, APIs, and other data communication scenarios.

**37.what is promises.**

Ans. In JavaScript, a Promise is an object that represents the eventual completion or failure of an asynchronous operation, and its resulting value. Promises are a way to handle asynchronous code more elegantly, avoiding callback hell and making it easier to reason about asynchronous tasks.

A Promise has three states:

1. \*\*Pending\*\*: The initial state; the promise is neither fulfilled nor rejected.

2. \*\*Fulfilled\*\*: The operation completed successfully, and the promise has a resulting value.

3. \*\*Rejected\*\*: The operation failed, and the promise has a reason for the failure.

The basic syntax for creating a Promise is as follows:

```javascript

const myPromise = new Promise((resolve, reject) => {

// Asynchronous operation

// If the operation is successful, call resolve with the result

// If the operation fails, call reject with the reason

});

```

You then use `.then()` to handle the fulfillment and `.catch()` to handle the rejection:

```javascript

myPromise

.then(result => {

// Handle fulfillment

console.log(result);

})

.catch(error => {

// Handle rejection

console.error(error);

});

```

Promises are commonly used in scenarios like making asynchronous requests (e.g., AJAX), reading files, and other tasks where the result isn't immediately available. They provide a cleaner and more structured way to work with asynchronous code compared to traditional callback patterns. Additionally, Promises can be chained using `.then()` and can be combined using methods like `Promise.all` and `Promise.race`.

**38.Write a program of promises and handle that promises also.**

Ans. // Function that returns a Promise to simulate an asynchronous operation

function simulateAsyncOperation(successful) {

return new Promise((resolve, reject) => {

setTimeout(() => {

if (successful) {

resolve("Operation completed successfully!");

} else {

reject("Operation failed!");

}

}, 2000); // Simulating a delay of 2 seconds

});

}

// Using the Promise

console.log("Start of the program");

const myPromise = simulateAsyncOperation(true);

myPromise

.then(result => {

console.log(result);

return "Additional data"; // Returning data to be used in the next then block

})

.then(additionalData => {

console.log(`Additional data: ${additionalData}`);

// Uncomment the line below to simulate a failure in the chain

// throw new Error("Error in the chain");

return "Another step completed!";

})

.then(finalResult => {

console.log(finalResult);

})

.catch(error => {

console.error(error);

})

.finally(() => {

console.log("End of the program");

});

**39. Use fetch method for calling an api** [**https://fakestoreapi.com/products**](https://fakestoreapi.com/products)

Ans. <!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Fetch API Example</title>

</head>

<body>

<script>

// Using the fetch method to make an API request

fetch('https://fakestoreapi.com/products')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('API Response:', data);

// Handle the data as needed

})

.catch(error => {

console.error('Error:', error);

});

</script>

</body>

</html>

**40. Display all the product from the api in your HTML page**

Ans.<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Display Products from API</title>

</head>

<body>

<div id=”products-container”></div>

<script>

Const productsContainer = document.getElementById(‘products-container’);

Fetch(‘https://fakestoreapi.com/products’)

.then(response => {

If (!response.ok) {

Throw new Error(`HTTP error! Status: ${response.status}`);

}

Return response.json();

})

.then(products => {

// Display the products in the HTML page

Products.forEach(product => {

Const productCard = document.createElement(‘div’);

productCard.innerHTML = `

<h3>${product.title}</h3>

<p>${product.description}</p>

<p>Price: $${product.price}</p>

<hr>

`;

productsContainer.appendChild(productCard);

});

})

.catch(error => {

Console.error(‘Error:’, error);

});

</script>

</body>

</html>

**41. What is JavaScript Output method?**

Ans. In JavaScript, there are several methods to output data or information to the user:

1. \*\*console.log\*\*: This method is used to log messages to the console, which is a tool available in most web browsers for debugging and development purposes. It's commonly used to print out values, variables, or messages for debugging purposes.

```javascript

console.log("Hello, world!");

```

2. \*\*alert\*\*: This method displays a dialog box with a message and an OK button. It's often used to provide information to the user or to ask for confirmation.

```javascript

alert("Hello, world!");

```

3. \*\*document.write\*\*: This method writes HTML expressions or JavaScript code to a document. It's rarely used in modern web development due to its potential to overwrite the entire document content.

```javascript

document.write("Hello, world!");

```

4. \*\*innerHTML\*\*: This property can be used to change the HTML content of an element. It's often used to dynamically update the content of a webpage.

```javascript

document.getElementById("output").innerHTML = "Hello, world!";

```

5. \*\*Return values\*\*: In functions or methods, you can use the return statement to output a value that can be used by the caller of the function.

```javascript

function greet() {

return "Hello, world!";

}

console.log(greet());

```

These methods provide different ways to output information in JavaScript, and the choice depends on the specific use case and context of your application.

**42.How to used JavaScript Output method?**

Ans. Using JavaScript output methods involves employing functions or properties that display information to the user or developers during the execution of your JavaScript code. Here are examples of how to use some of these methods:

### 1. `console.log`

```javascript

Let message = “Hello, world!”;

Console.log(message);

```

This will log “Hello, world!” to the browser console, which is useful for debugging and development.

### 2. `alert`

```javascript

Let message = “Hello, world!”;

Alert(message);

```

This will display an alert dialog box with the message “Hello, world!” to the user.

### 3. `document.write`

```javascript

Let message = “Hello, world!”;

Document.write(message);

```

This writes “Hello, world!” directly to the HTML document. Note that using `document.write` is less common in modern web development due to potential issues with overwriting content.

### 4. `innerHTML`

Assuming you have an HTML element with an id like this:

```html

<div id=”output”></div>

```

You can update its content using `innerHTML`:

```javascript

Let message = “Hello, world!”;

Document.getElementById(“output”).innerHTML = message;

```

### 5. Returning values

```javascript

Function greet() {

Return “Hello, world!”;

}

Let message = greet();

Console.log(message);

```

Here, the `greet` function returns a value, and we store that value in the variable `message`. We then log it to the console.

Choose the method that fits your specific use case and the desired way of communicating information to users or developers. Keep in mind that the appropriate method depends on the context of your application, whether it’s for debugging, user interaction, or dynamically updating HTML content.

**43.How to used JavaScript Events to do all examples?**

Ans. JavaScript events are used to trigger functions or actions in response to user interactions or other events in the browser. Here's how you can use JavaScript events to accomplish the examples provided earlier:

### 1. Using `console.log` with JavaScript Events:

You can attach a function to an event listener, such as `click`, to log a message to the console when a button is clicked:

```html

<button id="logButton">Log Message</button>

<script>

const logButton = document.getElementById("logButton");

logButton.addEventListener("click", function() {

console.log("Hello, world!");

});

</script>

```

### 2. Using `alert` with JavaScript Events:

Similarly, you can use an event listener to display an alert dialog box when a button is clicked:

```html

<button id="alertButton">Show Alert</button>

<script>

const alertButton = document.getElementById("alertButton");

alertButton.addEventListener("click", function() {

alert("Hello, world!");

});

</script>

```

### 3. Using `document.write` with JavaScript Events:

You generally don't use `document.write` with events, as it writes directly to the HTML document. Instead, you would typically use it during initial page load.

### 4. Using `innerHTML` with JavaScript Events:

You can update an element's content using `innerHTML` within an event listener:

```html

<div id="output"></div>

<button id="updateButton">Update Content</button>

<script>

const outputElement = document.getElementById("output");

const updateButton = document.getElementById("updateButton");

updateButton.addEventListener("click", function() {

outputElement.innerHTML = "Hello, world!";

});

</script>

```

### 5. Returning values with JavaScript Events:

You can trigger a function with an event listener, and that function can return a value that can then be used:

```html

<button id="returnButton">Return Message</button>

<script>

const returnButton = document.getElementById("returnButton");

returnButton.addEventListener("click", function() {

function greet() {

return "Hello, world!";

}

let message = greet();

console.log(message);

});

</script>

```

In each example, an event listener is added to a button element, and the specified action is executed when the button is clicked. This demonstrates how JavaScript events can be used to perform various actions in response to user interactions.