JavaScript Mastery

Web Dev Mastery

JavaScript Mastery

With 5 Projects & 2 GAME

{WDM}
Learn To Code

JavaScript Mastery

Web Dev Mastery

Environment SetUp :-

+

+

Js Environment

Code Editor

Browser

{WDM}
Learn To Code

First Js Code :-

Console.log is used to log (print) a message to the console

```
console.log("Web Dev Mastery / Suman");
```

Comments In Js :-

Part of Code which is not executed

```
// This is Single line comment

/* This is Single multi - line
        comment */
```

Variables In JS :-

Variables are just like containers to Store the data

- **Numbers** [{ -∞, +∞ } => Pincode , Mobile No., PAN, Bank Balance]
- **Strings** [‘a’ , ‘1’ , ‘@’ , “Suman” , “Web Dev Mastery”, “9843.2”]

Variables Rules :-

- Variable names are case sensitive; “a” & “A” is different.
- Only letters, digits, underscore (_) and $ is allowed (not even space).
- Only a letter, underscore (_) or $ should be 1st character.
- Reserved words cannot be variable names.

let, const & var :-

var : Variable can be re-declared & updated. A global scope variable.

let : Variable cannot be re-declared but can be updated. A block scope variable.

const : Variable cannot be re-declared or updated. A block scope variable.

Data Types In JS:-

JavaScript has 8 Datatypes

- String
- Number
- Bigint
- Boolean
- Undefined
- Null
- Symbol
- Object

JavaScript Types are Dynamic

Example

```
let x;    // Now x is undefined
x = 5;   // Now x is a Number
x = "John";    // Now x is a String
```

Note:- typeof - operator

OPerators In JS:-

Operators are Used to perform some operation on data

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Ternary Operator

OPerators In JS:-

Arithmetic Operators

+ , - , * , / , %

- Increment (+)
- Decrement (-)
- Multiply (*)
- Divide (/)
- Modulus (%)

OPerators In JS:-

Assignment Operators

1. =
2. +=
3. -=
4. *=
5. %=
6. **=

OPerators In JS:-

Comparison Operators

= = , ! = , = = = , ! = = , > , > = , < , < =

- Dobule Equal to (==) Only compare the value
- Tripple Equal to (===) Compare value & Data type
- Not Equal to (!=)
- Not Equql & Type (!==)
- Modulus (%)

Note :- == & === are not same

OPerators In JS:-

Logical Operators

- Logical AND &&
- Logical OR ||
- Logical NOT !

OPerators In JS:-

Ternary Operators

( ) ? ( ) : ( )

condition ? true output : false output

```
const  result =  marks  > 40 ? “pass” : “fail”
```

Template literal In JS:-

Template literal

```
const name =  "Suman"
const Id =  "22MCA10142"   // Using template literals
const greeting = `Hello, my name is ${name} and my Id ${Id}`;
console.log( greeting )
```

Conditional Statements In JS :-

If Statements

```
if (condition) {
// block of code to be executed if the condition is true
}
```

```
let greeting;
if (hour == 9) {
 greeting = "Good morning";
}
```

Conditional Statements In JS :-

If - else Statements

```
if (condition) { // block of code to be executed if the condition is true }
    else { // block of code to be executed if the condition is false }
```

```
let greeting;
if (hour == 9) { greeting = "Good morning"; }
else { greeting = "Good Afternoon"; }
```

Conditional Statements In JS :-

else - if Statements

```
if ( experience < 1 ) {
console.log( " Fresher Dev " );
 } else if ( experience > 2 ) {
console.log( " Senior Dev " );
} else { console.log ( " Berozgaar" );
  }
```

Conditional Statements In JS :-

Switch Statements

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break }
```

Loops In JS :-

Loops are used to execute a piece of code again & again

- for
- while
- do - while
- for - in
- for - of
- for each

Loops In JS :-

for Loop

A for loop is commonly used when the number of iterations is known .
It consists of three parts: initialization, condition, and final expression.

```
for ( initialization; condition; finalExpression )
{  // code to be executed }
```

Loops In JS :-

for Loop

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

Loops In JS :-

while Loop

Repeats a block of code as long as a specified condition is true

```
while (condition) {
// code runs while condition is true
}
```

Loops In JS :-

while Loop

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

Loops In JS :-

do while Loop

A do-while loop is similar to the while loop, but it checks the condition after executing the code block, ensuring the code runs at least once.

```
do {
// code to be executed
} while (condition);
```

Loops In JS :-

do while Loop

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

Function's In JS :-

Function's

A function is a block of code designed to perform a particular task.

It is executed when it is invoked or called.

Function's In JS :-

Function Definition

```
function function_Name( )
{
//do something
}
```

Function Calling

```
function_Name( )
```

Function's In JS :-

Function Declaration

```
function greet( name ) {
return `Hello, ${name}!`
}

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

Function's In JS :-

Arrow Function

```
const greet = ( name ) => {
return `Hello, ${name}!`
}

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

Function's In JS :-

Arrow Function

When the function body has only a single statement, you can omit the curly braces and the return keyword:

```
const greet = name  => `Hello, ${name}!`

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

Function's In JS :-

Function Global Scope

Index.js > ...

```
1   let globalVar = "I'm global!";
2
3   function display() {
4     console.log(globalVar); // Accessible inside the function
5   }
6
7   display(); // Output: I'm global!
8   console.log(globalVar); // Output: I'm global!
9
```

Function's In JS :-

Function Local Scope

Index.js > ...

```
1   function localScopeExample() {
2     let localVar = "I'm local!";
3     console.log(localVar); // Accessible inside the function
4   }
5
6   localScopeExample(); // Output: I'm local!
7   console.log(localVar); // Error: localVar is not defined (it's local to the function)
8
```

Function's In JS :-

Block Scope (with let and const)

Index.js > ...

```
1   if (true) {
2     let blockVar = "I'm block-scoped!";
3     console.log(blockVar); // Accessible inside the block
4   }
5
6   console.log(blockVar); // Error: blockVar is not defined (it's block-scoped)
7
```

Function's In JS :-

Callback Function

A callback function is a function that you pass as an argument to another function. It gets executed after a certain task is completed.

Callback Function

Index.js > ...

```js
// Define a callback function
const sayHello = (name) => {
  console.log(`Hello, ${name}!`);
};

// Define a function that takes a callback
const greetUser = (userName, callback) => {
  callback(userName); // Call the callback function with userName
};

// Call the function and pass the callback
greetUser("Suman", sayHello);
```

Object's In JS :-

An object in JavaScript is a collection of key-value pairs. The keys (properties) are strings (or symbols), and the values can be any data type (numbers, strings, arrays, functions, etc.).

Index.js > ...

```
1   const person = {
2     name: "Suman",
3     age: 25,
4     greet: function () {
5       return `Hello, my name is ${this.name}`;
6     }, };
7
8   console.log(person.name); // Output: Suman
9   console.log(person.greet()); // Output: Hello, my name is Suman
10
```

Object's In JS :-

Spread Operator (...)

Index.js > ...

```
1   const person = { name: "Suman", age: 25 };
2   const clone = { ...person };
3   console.log(clone); // Output: { name: 'Suman', age: 25 }
4
```

Object's In JS :-

Object Destructuring

Index.js > ...

```
1   const person = { Name: "Suman", age: 200, city: "Indore" };
2
3   // Destructuring the object
4   const { name, age, city } = person;
5
6   console.log(Name); // Output: Suman
7   console.log(age); // Output: 200
8   console.log(city); // Output: Indore
9
```

Array's In JS :-

- A array is a data structure that allows you to store multiple values in a single variable.

- Arrays are used to store lists of elements like numbers, strings, objects, and even other arrays.

- They are zero-indexed, meaning the first element has an index of 0, the second has an index of 1, and so on.

Array's In JS :-

Creating Array's

```
let fruits = [ 'apple', 'banana', 'orange' ];
let phones = [ 'apple', 'oneplus', 'samsung' ];
let score = [ 100, 89, 55, 0, 98, 78, 10 ];
let random = [ 91, "sony", 234.78, '@' ];
```

Array's In JS :-

Accessing Elements

```
let fruits = [ 'apple', 'banana', 'orange' ];

console.log ( fruits[0] ); // 'apple'

console.log ( fruits[2] ); // 'orange'
```

Array's In JS :-

Array Method's

1.) push(): Adds one or more elements to the end of the array.

```
Index.js > ...
1    let arr = [1, 2, 3];
2    arr.push(4);
3    // [1, 2, 3, 4]
4
```

Array's In JS :-

Array Method's

2.) pop(): Removes the last element from the array and returns that element.

Index.js > ...

```
1    let arr = [1, 2, 3];
2    arr.pop();
3    // [1, 2]
4
```

Array's In JS :-

Array Method's

3.) shift(): Removes the first element from the array and returns it.

Index.js > ...

```
1   let arr = [1, 2, 3];
2   arr.shift();
3   // [2, 3]
4
```

Array's In JS :-

Array Method's

4.) unshift(): Adds one or more elements to the beginning of the array.

Index.js > ...

```
1   let arr = [1, 2, 3];
2   arr.unshift(0);
3   // [0, 1, 2, 3]
4
```

Array's In JS :-

Array Method's

5.) length(): Returns the number of elements in the array.

Index.js > ...

```
1   let arr = [1, 2, 3];
2   console.log(arr.length);
3   // 3
4
```

Array's In JS :-

Array Method's

6.) find(): Returns the first element that satisfies the provided testing function.

Index.js > ...

```
1   let arr = [1, 2, 3, 4];
2   let found = arr.find((el) => el > 2);
3   // 3
4
```

Array's In JS :-

Array Method's

7.) includes(): Determines whether an array contains a certain value.

Index.js > ...

```
1   let arr = [1, 2, 3];
2   console.log(arr.includes(2));
3   // true
4
```

Array's In JS :-

Array Method's

8.) concat(): Merges two or more arrays and returns a new array.

Index.js > ...

```
1   let arr1 = [1, 2];
2   let arr2 = [3, 4];
3   let merged = arr1.concat(arr2);
4   // [1, 2, 3, 4]
5
```

Array's In JS :-

Array Method's

9.) join(): Joins all array elements into a string, with an optional separator.

Index.js > ...

```
1   let arr = ["apple", "banana", "cherry"];
2   let joined = arr.join(", ");
3   // 'apple, banana, cherry'
4
```

Array's In JS :-

Array Method's

10.) splice(): Adds or removes elements from the array.

Index.js > ...

```
1   let arr = [1, 2, 3, 4];
2   arr.splice(2, 1, "a");
3   // [1, 2, 'a', 4] (removed 1 element at index 2 and added 'a')
4
```

Array's In JS :-

Array Method's

11.) slice(): Returns a shallow copy of a portion of an array.

Index.js > ...

```
1   let arr = [1, 2, 3, 4, 5];
2   let sliced = arr.slice(1, 3);
3   // [2, 3]
4
```

Array's In JS :-

Array Method's

12.) sort(): Sorts the elements of the array (alphabetical by default, can be customized).

Index.js > ...

```
1   let arr = [3, 1, 4, 2];
2   arr.sort((a, b) => a - b);
3   // [1, 2, 3, 4]
4
```

Array's In JS :-

Array Method's

13.) findIndex(): Returns the index of the first element that satisfies a test.

Index.js > ...

```
1   let arr = [5, 12, 8];
2   let index = arr.findIndex((el) => el > 10);
3   // 1
4
```

Array's In JS :-

Array Method's

14.) from(): Creates an array from an array-like or iterable object

Index.js > ...

```
1   let str = "hello";
2   let arr = Array.from(str);
3   // ['h', 'e', 'l', 'l', 'o']
4
```

Array's In JS :-

Array Method's

15.) isArray(): Checks if the given value is an array.

Index.js

```
1   console.log(Array.isArray([1, 2, 3]));
2   // true
3   console.log(Array.isArray("hello"));
4   // false
5
```

Array's In JS :-

High Order Array Method's

1.) map(): Creates and return new array by applying a function to each element of the original array.

Index.js > ...

```
1   let arr = [1, 2, 3];
2   let doubled = arr.map((el) => el * 2);
3   console.log(doubled);
4   // [2, 4, 6]
5
```

Array's In JS :-

High Order Array Method's

2.) filter(): Creates and returns a new array with elements that pass a specified test condition.

Index.js > ...

```
1   let arr = [1, 2, 3, 4];
2   let evens = arr.filter((el) => el % 2 === 0);
3   console.log(evens);
4   // [2, 4]
5
```

Array's In JS :-

High Order Array Method's

3.) reduce(): Reduces an array to a single value by applying a function to each element.

Index.js > ...

```
1   let arr = [1, 2, 3, 4];
2   let sum = arr.reduce((acc, el) => acc + el, 0);
3   console.log(sum);
4   // 10
5
```

Advance Loop's In JS :-

for...in Iterates over the keys (properties) of an object or the indices of an array.

Index.js > ...

```
1   let obj = { a: 1, b: 2, c: 3 };
2   for (let key in obj) {
3     console.log(key, obj[key]);
4   }
5   // Output:
6   // a 1
7   // b 2
8   // c 3
9
```

Advance Loop's In JS :-

for...of Iterates over the values of iterable objects like arrays, strings, Maps, etc.

Index.js > ...

```
1   let arr = [10, 20, 30];
2   for (let value of arr) {
3     console.log(value);
4   }
5   // Output:
6   // 10
7   // 20
8   // 30
9
```

Advance Loop's In JS :-

forEach()

Executes a function once for each element in an array (cannot be used to break the loop).

Index.js > ...

```
1   let arr = [10, 20, 30];
2   arr.forEach((value, index) => {
3     console.log(index, value);
4   });
5   // Output:
6   // 0 10
7   // 1 20
8   // 2 30
9
```

String's In JS :-

Strings are a sequence of characters used for representing text.

Declaring a String

- Double quotes (" ")
- Single quotes (' ')
- Backticks (for template literals) ``

String's In JS :-

Example :-

Index.js > ...

```
1   let str1 = "Hello, World!";  // using double quotes
2   let str2 = 'Hello, JavaScript!';  // using single quotes
3   let str3 = `Hello, Universe!`;  // using backticks (template literals)
4
```

String's In JS :-

Method's

- length – Returns the number of characters in the string.
- toUpperCase() – Converts the string to uppercase.
- toLowerCase() – Converts the string to lowercase.
- includes() – Checks if the string contains a specific substring.
- indexOf() – Returns the index of the first occurrence of a substring
- trim() – Removes whitespace from both ends of the string.

String's In JS :-

Method's

- substring(start, end) – Extracts a substring between two specified indices.
- slice(start, end) – Extracts a portion of the string, supporting negative indices.
- replace(old, new) – Replaces a specified substring with another substring.
- split(separator) – Splits the string into an array based on a separator.
- charAt(index) – Returns the character at the specified index.

Date & Time In JS :-

Date Object

Index.js > ...

```
1   let currentDate = new Date();
2   console.log(currentDate); // Shows the current date and time
3
```

Date & Time In JS :-

Method's

- **getFullYear():** Returns the year (e.g., 2024).
- **getMonth():** Returns the month (0-11).
- **getDate():** Returns the day of the month (1-31).
- **getHours():** Returns the hour (0-23).
- **getMinutes():** Returns the minutes (0-59).
- **getSeconds():** Returns the seconds (0-59).

Date & Time In JS :-

Example:

Index.js > ...

```
1   let now = new Date();
2   console.log(
3     `Year: ${now.getFullYear()}, Month: ${
4       now.getMonth() + 1
5     }, Day: ${now.getDate()}`
6   );
7
```

setInterval() In JS :-

setInterval()

setInterval() is used to execute a function repeatedly after a given interval of time (in milliseconds).

Index.js > ...

```
1   let intervalId = setInterval(function () {
2     console.log("This runs every 2 seconds");
3   }, 2000); // Repeats every 2000ms (2 seconds)
4
```

clearInterval() In JS :-

clearInterval

Stopping setInterval(): Use clearInterval(intervalId) to stop the interval.

Index.js

```
1   clearInterval(intervalId); // Stops the repeated execution
2
```

setTimeout() In JS :-

setTimeout()

setTimeout() :- is used to execute a function after a specified delay (in milliseconds), but it only runs once.

Index.js > ...

```
1   setTimeout(function () {
2     console.log("This runs once after 3 seconds");
3   }, 3000); // Executes after 3000ms (3 seconds)
4
```

Example combining

Index.js > ...

```js
function displayTime() {
  let now = new Date();
  console.log(
    `Time: ${now.getHours()}:${now.getMinutes()}:${now.getSeconds()}`)}

// Print the time every second using setInterval
let intervalId = setInterval(displayTime, 1000);

// Stop displaying the time after 10 seconds using setTimeout
setTimeout(function () {
  clearInterval(intervalId);
  console.log("Stopped displaying time after 10 seconds.");
}, 10000);
```

Sync & Async JavaScript :-

Synchronous & Asynchronous are two different ways that JavaScript executes code. Understanding these concepts is crucial for managing tasks, especially in a web environment where you deal with user interactions, network requests, and more.

Sync & Async JavaScript :-

Synchronous JavaScript

Definition: In synchronous execution, code runs line by line, and each line must finish executing before the next one starts. This can lead to delays if a task takes a long time (e.g., fetching data).

Index.js

```
1   console.log("Start"); // 1
2   console.log("This runs immediately"); // 2
3   console.log("End"); // 3
4
```

Sync & Async JavaScript :-

Asynchronous JavaScript

Definition: In asynchronous execution, certain operations can be initiated and will run in the background, allowing the rest of the code to continue executing without waiting for the task to finish.

Characteristics:

- Non-blocking: Other code can run while waiting for an operation (like a network request) to complete.

JavaScript Mastery

Web Dev Mastery

Sync & Async JavaScript :-

Asynchronous JavaScript

Index.js > ...

```
1   console.log("Asynchronous Code");
2
3   setTimeout(() => {
4     // Simulate a delay
5     const data = { user: "Suman" };
6     console.log(`Hello: ${data.user}`);
7   }, 2000); // 2 seconds delay
8
9   console.log("This runs before { Hello: Suman }");
10
```

{WDM} Learn To Code

Sync & Async JavaScript :-

Key Differences :-

Synchronous	Asynchronous
Executes line by line.	Executes tasks in the background, allowing other code to run.
Blocks further execution until the current task is complete.	Does not block; can continue running while waiting for tasks to finish.
Easier to read and understand for simple tasks.	More complex but essential for handling tasks like API calls, timers, etc.

JavaScript Mastery

Web Dev Mastery

DOM In JavaScript :-

DOM Introduction:-

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a webpage as a tree of objects, allowing programming languages (like JavaScript) to access, modify, and manipulate the document's content, structure, and style.

WDM
Learn To Code

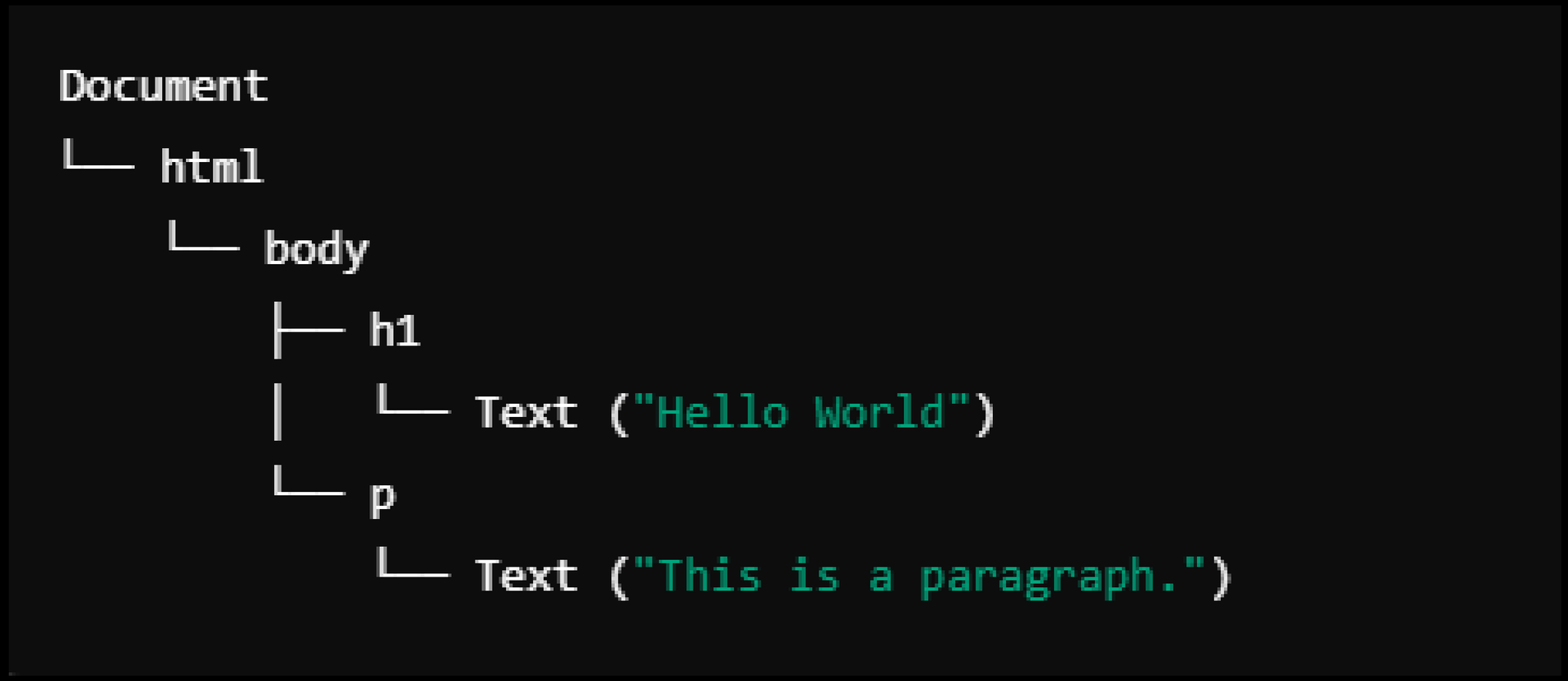
DOM In JavaScript :-

Code Snippet :-

```
<html>
    <body>
        <h1>Hello World</h1>
        <p>This is a paragraph.</p>
    </body>
</html>
```

DOM In JavaScript :-

DOM Tree For Above Code Snippet :-



DOM In JavaScript :-

Accessing HTML Elements:

- getElementById(“myId”)
- getElementsByClassName(“myClass”)
- getElementsByTagName(“h1”)
- querySelector(“div”) *// returns first element*
- querySelectorAll(“div”) *// returns a NodeList*

DOM Manipulation :-

DOM Element Properties:

- textContent : Gets/sets the text content (no HTML).
- innerHTML : Gets/sets the HTML content (with tags).
- innerText : Gets/sets visible text (ignores hidden).
- style : Accesses inline styles.
- className : Gets/sets class name(s).
- tagName : Returns the element's tag name.
- src : Gets/sets image source.

DOM Manipulation :-

Creating Elements:

- createElement(tagName): Creates a new element (e.g., div).

DOM Attributes:

- setAttribute(attribute, value): Sets an attribute's value.
- getAttribute(attribute): Gets an attribute's value.
- removeAttribute(attribute): Removes an attribute.

DOM Manipulation :-

Insert / Delete Elements:

```
let div = document.createElement("div")

const node = document.getElementById('mainNode');
```

- `node.append(div)` *// adds at the end of node (inside).*
- `node.prepend(div)` *// adds at the start of node (inside)).*
- `node.before(div)` *// adds before the node (outside) .*
- `node.after(div)` *// adds after the node (outside).*
- `node.remove(div)` *// removes the node*

Events In JavaScript :-

An event is an action or occurrence that happens in the browser, usually as a result of user interaction or the browser's system processes.

Types of Events:

- Mouse events : (click , dblclick , mouseover , mouseout)
- Keyboard events : (keypress , keyup , keydown)
- Form events : (submit , change , focus)
- Window events : (load , resize , scroll)

Events In JavaScript :-

Event Handling :

This is a process of responding to user interactions or occurrences in the browser, such as clicks, key presses, or form submissions.

Example:

```
function handleClick() {
alert("Button clicked!");
}
```

Ways to Assign Event Handlers :-

1.) HTML Attribute :

```
<button onclick="handleClick()">Click Me!</button>
```

2.) Inline JavaScript :

```
document.getElementById("myButton").onclick = handleClick;
```

Event Listeners :-

An event listener is a method that listens for a specific event to happen on a particular element.

Syntax: `element.addEventListener("event", eventHandler)`

- element : The DOM element you want to attach the listener to.
- event : The type of event (e.g., click, submit, keydown).
- eventHandler : function to be executed when the event occurs.

Event Listeners :-

Example :

Index.js > ...

```
1   const button = document.getElementById("myButton");
2
3   // Add an event listener for 'click' event
4   button.addEventListener("click", handleClick);
5
```

Event Object :-

- When an event occurs, an event object is automatically created and passed to the event handler.
- This object contains useful information about the event, such as the type of event, the target element, and the position of the mouse.

Example :- e.target , e.type , e.clientX , e.clientY

Event Object :-

Example :-

Index.js > ...

```
1   button.addEventListener("click", function (event) {
2     console.log(event.type); // Outputs: 'click'
3     console.log(event.target); // Outputs: the button element
4   });
5
```

BOM In JavaScript :-

The BOM (Browser Object Model) is a collection of objects that allow JavaScript to interact with the browser.

BOM components :

- Window Object
- Location Object
- Alert, Prompt, Confirm

BOM In JavaScript :-

Window Object & Method's :

- window.open() : Opens a new tab/window.
- window.close() : Closes the current window.
- window.scrollTo() : Scrolls the window to a position.
- window.setTimeout() : Delays code execution.
- window.setInterval() : Repeats code at intervals.

BOM In JavaScript :-

Location Object & Method's :

- location.href : Gets or sets the current URL.
- location.reload() : Reloads the page.
- location.assign() : Loads a new URL.
- location.replace() : Replaces the current page.
- location.pathname : Gets the URL path.

BOM In JavaScript :-

Alert, Prompt, Confirm:

- alert() : Shows a message box.
- prompt() : Asks for user input.
- confirm() : Asks for confirmation (OK/Cancel).

Promises In JavaScript :-

A Promise in JavaScript is an object representing the eventual result (success/failure) of an asynchronous operation.

States of a Promise:

- **Pending :** Initial state, operation hasn't completed.
- **Fulfilled :** Operation succeeded, giving a resolved value.
- **Rejected :** Operation failed, providing a reason (error).

Promises In JavaScript :-

Basic Syntax:

Index.js > ...

```
1   let promise = new Promise((resolve, reject) => {
2     // async code
3     if (success) resolve("Resolved data");  // Fulfilled
4     else reject("Error message");           // Rejected
5   });
6
```

Promises In JavaScript :-

Handling Promise Results :-

- **.then() for Fulfilled Promises:**

  .then() runs if the promise is fulfilled.

- **Syntax:** *promise.then(result => { /* code */ })*
- **.catch() for Rejected Promises:**

  .catch() runs if the promise is rejected.

- **Syntax:** *promise.catch(error => { /* code */ })*

Sync & Async In JavaScript :-

Synchronous :

Executes tasks one at a time, blocking further execution until the current task finishes.

Asynchronous :

Allows tasks to run concurrently, enabling other code to execute without waiting for the current task to complete.

aync await In JavaScript :-

async :

Used to define a function that runs asynchronously and automatically returns a promise. Allows the function to use **await** for handling promises more cleanly.

await :

Pauses execution inside an **async** function until a promise resolves, then returns the resolved value. Makes asynchronous code look synchronous for better readability.

JavaScript Mastery

Web Dev Mastery

Fetch Data From API :-

Index.js > fetchData

```
async function fetchData() {
  try {
    const response = await fetch("https://api.example.com/data");
    if (!response.ok) {
      throw new Error("Network response was not ok");
    }
    const data = await response.json(); // Wait for JSON parsing
    console.log(data); // Use the fetched data
  } catch (error) {
    console.error("There was a problem with the fetch operation:", error);
  } }
```

{WDM} Learn To Code

JavaScript Mastery

Web Dev Mastery

Thanks for Watching

Please leave a Like & Comment

{WDM}
Learn To Code