Java Script

- JavaScript is the most popular programming language which is used to create interactive webpages
- JavaScript is the client-server side scripting language which is used to create dynamic web pages
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.
- Current version of JS is ES6 (introduced after 2015)

Characteristics of JavaScript

- **Client-side scripting language :** The code is readable analyzable and executable by browser itself
- **Interpreted Language :** JavaScript is an interpreted language, meaning it is executed line by line at runtime.
- **Dynamically types Language :** JavaScript is dynamically-typed, which means you don't need to specify the data type of a variable when declaring it. The type is determined at runtime.
- **Weakly Typed Language :** Semicolon is not mandatory at all the time
- **Synchronous in nature :** It will execute line by line ; we can make it as asynchronous

Advantages of JS

- Easy to learn and implement
- Reduces server load
- Efficient performance
- Regular updates
- Platform independent
- It has more frameworks and libraries

Environmental Setup

- We can run JavaScript code in two environments.
  - Browser
  - Node.js
- Browser understand the JS through JS engine and Every browser consists of JavaScript engine which helps to run JS code

Browser	JS engine
Chrome	**V8**
Mozilla Firefox	**Spider monkey**
Microsoft Edge	**Chakra**
Safari	**JavaScript Core Webkit**

Output Methods

- **Console.log("hello")**
  - It will print the output in console window
- **Document.write("Hello")**
  - It will print the output in user interface
- **Document.writeln("Hello")**
  - It will print the output in user interface

Tokens

Tokens are Building blocks of every programming language

1. **Keywords :** Predefined words whose task is already predefined
2. **Identifiers :** The name given for the variables ,arrays , objects etc..
3. **Literals :** The values written by developer

Variables

- A variable is a container that stores a value. This is very similar to the container used to store rice, water.
- In JavaScript We have 3 keywords to declare variables(var , let , const)
- Syntax To declare variable:
  - var/let/const identifier;
  - var/let/const identifier=value;

Rules for choosing variable names:

- Letters, digits, underscores and $ sign allowed.
- Must begin with $, _ or a letter.
- JavaScript reserved words cannot be used as a variable name.
- Saurabh and saurabh are different variables(case sensitive).

Var , let and const

	var	Let	Const
Declaration	**yes**	**yes**	**Declaration with initialization**
Initialization	**Yes**	**Yes**	**No**
Reinitialization	**Yes**	**Yes**	**No**
Redeclaration	**Yes**	**No**	**No**
Scope	**Global scope**	**Script scope**	**Script scope**
Hoisting	**Yes**	**Yes(TDZ)**	**Yes(TDZ)**

Hoisting

- Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).
- In JavaScript, a variable can be declared after it has been used.
- Example:

```
console.log(myVar); // Outputs: undefined
var myVar = 42;
```

The let and const Keywords

- Variables defined with **let** and **const** are hoisted to the top of the block, but not ***initialized***.
- **Meaning:** The block of code is aware of the variable, but it cannot be used until it has been declared.
- Using a let variable before it is declared will result in a ReferenceError.
- The variable is in a "temporal dead zone" from the start of the block until it is declared:
- A temporal dead zone (TDZ) is the area of a block where a variable is inaccessible until the moment the computer completely initializes it with a value.

Datatypes in JS

Datatype defines the type of data to be stored in a particular memory allocation

JavaScript has 7 Datatypes(primitive)

Primitive data types are a set of basic data types in JavaScript.

1. String.
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol

Non-primitive

Objects is a non-primitive datatype in JavaScript

- A function
- An object
- An array

Object can be created as follows:

```
const item = {
    name : "led bulb",
    price : "150"
}
```

- **String :** We can declare the string in JS in 3 ways
- **Example :** 'hello' ,"hello" ,`Hello everyone`
- **Number :** Integer and floating-point numbers
- **Boolean :** True or false
- **Bigint :** large amount of data
- **Undefined :** It represents the value of an uninitialized values
- **Null :** It is empty variable and for future use
- **Symbol :** It will create function (mostly used in MongoDB)

Operators in JS

Operators are Predefined symbols used to perform particular task

- **Types of operators**

- Arithmetic Operators : +,-,*,/,%,++,--
- Assignment Operators : +=,-=,*=,/=,%=,=
- Comparison Operators : >,<, !=,>=,<=,==,===
- Logical Operators : && , || , !
- Ternary Operators : (condition) ? True stmt : false stmt

Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

Loops in JS

Loops can execute a block of code a number of times

while loop : Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body

Do while loop : It is more like a while statement, except that it tests the condition at the end of the loop body.

For loop : Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable

For in loop : Loops through the keys of an object

For of loop : Loops through the values of an object.

Functions In JS

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- With functions you can reuse code
- You can write code that can be used many times.
- You can use the same code with different arguments, to produce different results.

Syntax :

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: (*parameter1, parameter2, ...*)
- The code to be executed, by the function, is placed inside curly brackets: {}

Types of Functions in JS

1. Anonymous function
2. Named function
3. Function with expression
4. Nested function
5. Immediate invoking function
6. Arrow function
7. Higher order function
8. Callback function

1. Anonymous function

A function without name is known as Anonymous function

Syntax :

```
function(parameters) {
    // function body
}
```

2. Named Function

A function with name is called as named function

Syntax :

```
function functionName(parameters) {
    // function body
}
```

3. Function with expression

- It is the way to execute the anonymous function
- Passing whole anonymous function as a value to a variable is known as function with expression.
- The function which is passed as a value is known as first class function

EX :

```
let x = function () {
    //block of code
}
x();
```

4. Immediate invoking function expression (IIFE)

A function which is called immediately as soon as the function declared is known as IIFE

We can invoke anonymous function by using IIFE

Example:

```
(function () {
    console.log(“Hello”);
})();
```

5. Nested function

- A function which is declared inside another function is known as nested function.
- Nested functions are unidirectional i.e., We can access parent function properties in child function but vice-versa is not possible.
- The ability of JS engine to search a variable in the outer scope when it is not available in the local scope is known as lexical scoping or scope chaining.
- Whenever the child function needs a property from parent function the closure will be formed and it consists of only required data.
- A closure is a feature of JavaScript that allows inner functions to access the outer scope of a function. Closure helps in binding a function to its outer boundary and is created automatically whenever a function is created.

```
function parent(){
        let a = 10;
        function child() {
            let b = 20;
            console.log(a + b);
        }
        Child();
    }
    Parent();
```

JavaScript currying

Calling a child function along with parent by using one more parenthesis is known as java script currying

Example:

```
function parent(){
        let a = 10;
        function child() {
            let b = 20;
            console.log(a + b);
        }
        return child;
    }
    parent()();     //JavaScript currying
```

6. Arrow function

- It was introduced in ES6 version of JS.
- The main purpose of using arrow function is to reduce the syntax.
- If function has only one statement then block is optional.
- It is mandatory to use block if function consists of multiple lines of code or if we use return keyword in function.

Example:

```
    let x = (a, b) => console.log(a + b);
    let y = (a, b) => { return a + b };
```

7. Higher order function(HOF)

A function which accepts a function as an argument is known as HOF

It is used to perform multiple operations with different values.

Example:

```
function hof(a, b, task) {
    let res = task(a, b);
    return res;
};
let add = hof(10, 20, function (x, y) {
    return x + y;
});
let mul = hof(10, 20, function (x, y) {
    return x * y;
});
Console.log(add());
Console.log(mul());
```

8. Callback function

A function which is passed as an argument to another function is known as callback function.

The function is invoked in the outer function to complete an action

Example :

```
function first() {
    Console.log(“first”);
}
function second() {
    Console.log(“third”);
}
function third(callback) {
    Console.log(“second”);
    Callback()
}
first();
third(second);
```

Strings

Strings are used to store and manipulate text. String can be created using the following syntax:

```
let name = "saurabh"
```

String can also be created using single quotes

```
let name = ‘saurabh’
```

Template literals

Template literals use backticks instead of quotes to define a string.

```
let name = `saurabh`
```

With template literals, it is possible to use both single as well as double quotes inside a string.

```
let name = `The name "is" Saurabh`
```

We can directly insert variables in template literal. This is called string interpolation.

```
let name = `The name ${name}`  //----> Print this is Saurabh
```

Escape sequence characters

If you try to print the following string, JavaScript will misunderstand it

```
let name = 'Adam D'angelo`
```

we can use single quote escape sequence to solve the problem

```
let name = `adam D\'Angeo`
```

Similarly, we can use ‘ \’ inside a string with double quotes.

Other escape sequence characters are as follows

\n → New line

\t → Tab

\r → Carraige return

String properties and methods

Properties:

1. **length**
   Returns the number of characters in a string.

```
let str = "Hello, World!";
console.log(str.length); // Output: 13
```

Methods:

1. **charAt(index)**
   Returns the character at the specified index.

```
let str = "Hello";
console.log(str.charAt(1)); // Output: "e"
```

2. **charCodeAt(index)**
   Returns the Unicode of the character at the specified index.

```
let str = "Hello";
console.log(str.charCodeAt(1)); // Output: 101 (Unicode for 'e')
```

3. **concat(string2, string3, ..., stringN)**
   Combines two or more strings and returns a new string.

```
let str1 = "Hello";
let str2 = "World";
console.log(str1.concat(" ", str2)); // Output: "Hello World"
```

4. **includes(substring, startPosition)**
   Checks if a string contains a specified substring, starting from an optional position.

```
let str = "Hello, World!";
console.log(str.includes("World")); // Output: true
```

5. **indexOf(substring, startPosition)**
   Returns the index of the first occurrence of a substring, starting from an optional position. Returns -1 if not found.

```
let str = "Hello, World!";
console.log(str.indexOf("World")); // Output: 7
```

6. **lastIndexOf(substring, startPosition)**
   Returns the index of the last occurrence of a substring.

```
let str = "Hello, World! Hello again!";
console.log(str.lastIndexOf("Hello")); // Output: 13
```

7. **slice(startIndex, endIndex)**
   Extracts a part of a string and returns a new string, without modifying the original string.

```
let str = "Hello, World!";
console.log(str.slice(0, 5)); // Output: "Hello"
```

8. **substring(startIndex, endIndex)**
   Similar to slice(), but does not accept negative indices.

```
let str = "Hello, World!";
console.log(str.substring(0, 5)); // Output: "Hello"
```

9. **toLowerCase()**
   Converts a string to lowercase.

```
let str = "Hello, World!";
console.log(str.toLowerCase()); // Output: "hello, world!"
```

10. **toUpperCase()**
    Converts a string to uppercase.

```
let str = "Hello, World!";
console.log(str.toUpperCase()); // Output: "HELLO, WORLD!"
```

11. **trim()**
    Removes whitespace from both ends of a string.

```
let str = "   Hello, World!   ";
console.log(str.trim()); // Output: "Hello, World!"
```

12. **replace(searchValue, newValue)**
    Replaces the first occurrence of a substring with a new value.

```
let str = "Hello, World!";
console.log(str.replace("World", "JavaScript")); // Output: "Hello, JavaScript!"
```

13. **split(separator, limit)**
    Splits a string into an array of substrings, based on a separator.

```
let str = "Hello, World!";
let arr = str.split(", ");
console.log(arr); // Output: ["Hello", "World!"]
```

14. **repeat(count)**
    Repeats a string a specified number of times.

```
let str = "Hello";
console.log(str.repeat(3)); // Output: "HelloHelloHello"
```

15. **startsWith(substring, startPosition)**
    Checks if a string starts with a specific substring.

```
let str = "Hello, World!";
console.log(str.startsWith("Hello")); // Output: true
```

16. **endsWith(substring, length)**
    Checks if a string ends with a specific substring.

```
let str = "Hello, World!";
console.log(str.endsWith("World!")); // Output: true
```

JAVASCRIPT ARRAYS

ARRAYS:-

Array is collection of different elements.

Array is heterogenous in nature.

- In JavaScript we can create array in 2 ways.
    1. By array literal
    2. By using an Array constructor (using new keyword)

1. JavaScript array literal:-

- The syntax of creating array using array literal is:

```
var arrayname = [value1, value2.....valueN];
```

- As you can see, values are contained inside [ ] and separated by , (comma).
- The .length property returns the length of an array.

Ex:-

```
var emp = ["Sonoo", "Vimal", "Ratan"]; for (i = 0; i < emp.length; i++) {
    document.write(emp[i] + "<br/>"); // Sonoo  Vimal  Ratan
}
```

2. JavaScript array constructor (new keyword):-

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

Ex:-

```
var emp = new Array("Jai", "Vijay", "Smith");
for (i = 0; i < emp.length; i++) {
    document.write(emp[i] + "<br>"); // Jai Vijay Smith
}
```

Java script Array methods

- **push() :** It will insert an element of an array at the end
- **unshift()** : It will insert an element of an array at the first
- **pop() :** It will remove elements of array from the end
- **shift() :** It will remove elements of array from the start
- **indexOf()** : It will return an index of particular element
- **Includes()** : It will check whether the particular element is present in array or not.
- **At() :** It will return the element which is present in particular index.
- **Slice()** : It will give slice of an array and it will not affect the original array.

- **Splice() :** It is used to add or remove elements from an array
- **Join() :** It is used to join all elements of an array into a string.
- **Concat()** : It is used to join/concat two or more arrays.
- **toString()** : It converts all the elements in an array into a single string and returns that string.
- **Split() :** It is used to split a string into an array.
- **Reverse()** : It is used to reverse an array.
- **Array.from()** : It will convert string into an array

Filter() , map() , reduce()

1. Filter() :

filter() is a HOF which will check a particular condition for each element in the original array. If the element satisfies the condition the element will be pushed to the new array.

Syntax :

```
arr.filter((ele, index, arr) => {
    return condition
})
```

Example :

```
let numbers = [1, 2, 3, 4, 5];
let evenNo = numbers.filter((x) => x % 2 === 0);
console.log(evenNo); //[2, 4]
```

- **callback:** This is the function that is used to test each element in the array. It takes three arguments:
    - **element:** The current element being processed in the array.
    - **index (optional):** The index of the current element being processed.
    - **array (optional):** The array filter was called upon.

2. Map() :

The map() method in JavaScript is used to create a new array by applying a function to every element in an existing array. It does not modify the original array. Instead, it returns a new array with the modified elements.

EX:

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.Map(number => number * 2);
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

3. Reduce() :

The reduce() is a HOF which will return a single value from the original array. if no initial value is given, the accumulator will be assigned with the first value of the array

SYNTAX:-

```
arr.reduce((acc, ele, index, arr) => {
    //statements
}, init)
```

Example: -

```
let arr = [1, 2, 3, 4, 5];
let sum = arr.reduce((accumulator, currentValue, index, arr) => {
    console.log(accumulator, currentValue, index);
    return accumulator + currentValue
}, 100);
console.log(sum);//115
```

JavaScript in the Browser

JavaScript was initially created to make web pages alive. JS can be written right in a web page's HTML to make it interactive.

The browser has an embedded engine called JavaScript Engine or the JavaScript runtime.

JavaScript ability in the browser is very limited to protect the user's safety. For Example, a Webpage on **google** access **Amazon.in** and steal information from there.

Developer's tools

Every browser has some developer tools which makes a developer's life a lot easier.

Script tag

The script tag is used to insert JavaScript into an HTML page.

The script tag can be used to insert external or internal scripts.

```
<script>
    alert("hello")
</script>
```

The benefit of a separate JavaScript file is that the browser will download it and store it in its cache memory.

Console Object Methods

The Console object has several methods, log being one of them. Some of them are as follows:

1. Assert() : used to assert a condition.
2. Clear() : clears the console
3. Log() : outputs a message to the console
4. Table() : displays a tabular data
5. Warn() : used for warnings
6. Error() : used for errors
7. Info() : used for special information.

BOM and DOM

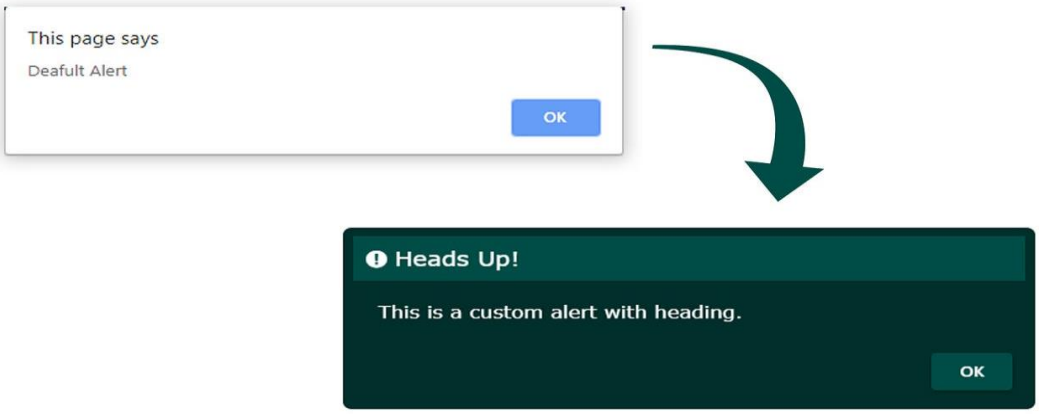
BROWSER OBJECT MODEL AND DOCUMENT OBJECT MODEL

BOM(Browser Object Model)

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- "BOM" stands for Browser Object Model. It represents the objects that a browser provides to JavaScript to interact with the browser itself.

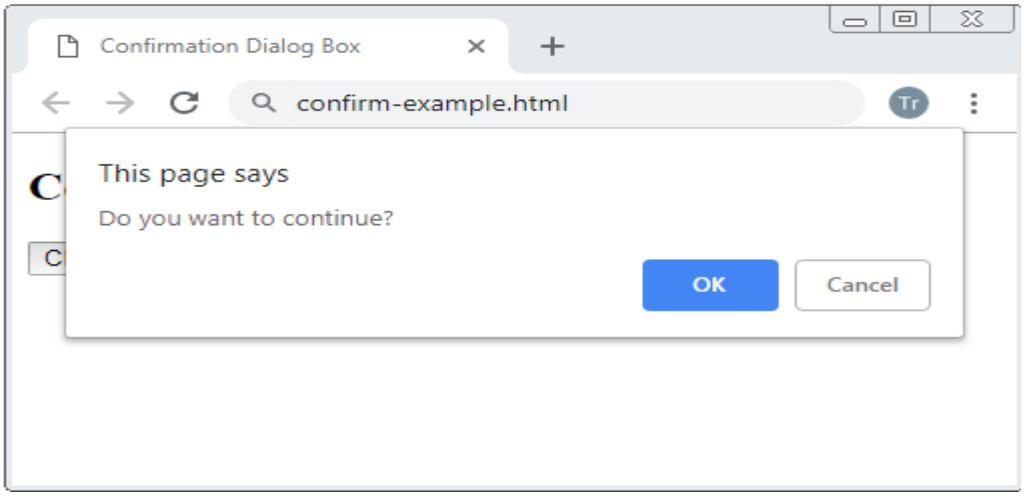
JavaScript Popup Boxes

- JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.
- **Alert Box**
- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.



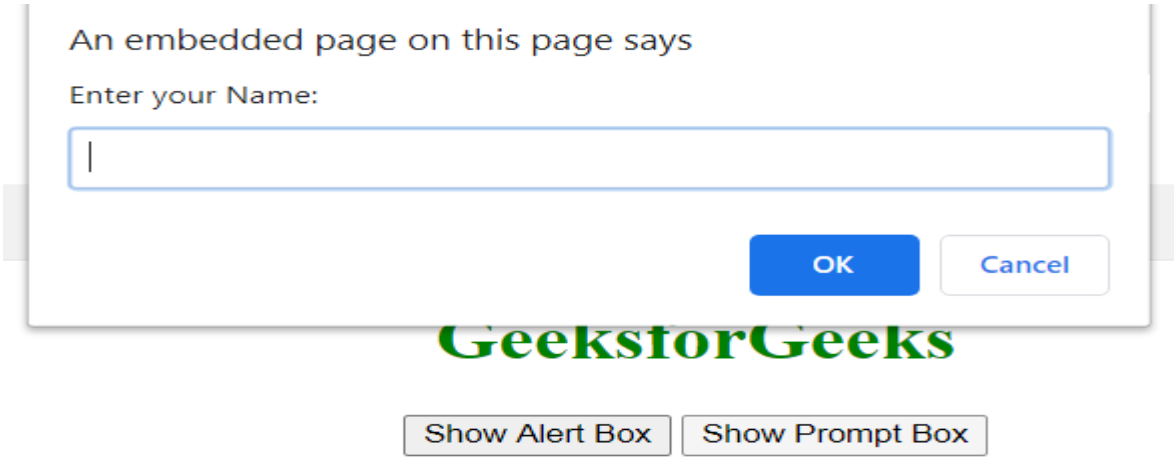
Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.



Prompt Box :

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.



DOM(Document Object Model)

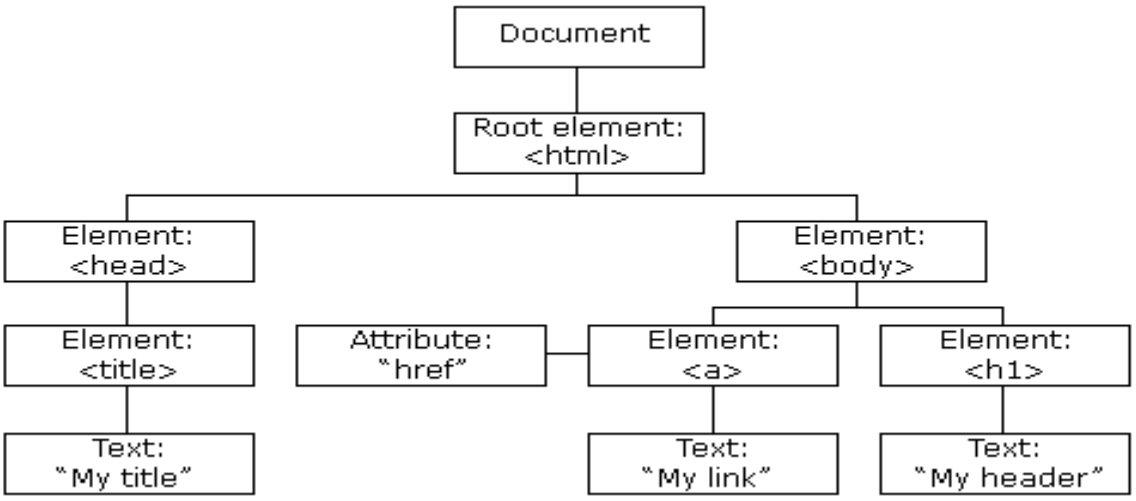
- In JavaScript, the DOM (Document Object Model) is a programming interface for web documents. It represents the structure of a document as a tree-like model where each node is an object representing a part of the document, such as elements, attributes, and text.
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:
- **DOM** tree refers to the HTML page where a;; the nodes are objects. There can be 3 main types of nodes in the DOM tree:
    1) **Text node**
    2) **Element node**
    3) **Comment node**

In an HTML page, <html> is at the root and <head> and <body> are its children.

A **text node** is always a leaf of the tree.

Auto correction

If an erroneous HTML is encountered by the browser, it tends to correct it, for example, if we put something after the body, it is automatically moved inside the body. Another example is <table> tag which is must contain <tbody>.



Note:

Document body can sometimes be null if the JavaScript is written before the body tag.

Children of an element

Direct as well as deeply nested elements of an element are called its children.

Child Nodes: Elements that are direct children. E.g. head and body are children of <html>.

Descendant nodes : All nested elements, children, their children and so on.

FirstChild , LastChild & ChildNodes

Element.fisrtChild	first child element
Element.lastChild	last child element
Element.childNodes	all Child nodes

Notes on DOM Collection

- They are read-only.
- They are live elements. Childnodes (references) will automatically update if childnodes of item is changed.
- They are Iterable using for..of loop

Siblings and the parent

Siblings are nodes that ae children of the same parent

EX:

- <head> and <body> are the siblings. Siblings have same parent. In the above example its html.
- <body> is said to be the “next” or “right” sibling of <head>, <head> is said to be “previous” or the “left” sibling of <body>.

Searching the DOM

DOM navigation properties are helpful when the elements are close to each other. If they are not close to each other, we have some more methods to search the DOM.

Methods used to target HTML elements in JavaScript file

- **getElementById(id):** This method allows you to retrieve an element from the document by its unique id.
- **getElementsByClassName(className):** This method returns a collection of all elements in the document with a specified class name.
- **getElementsByTagName(tagName):** Returns a collection of elements with the specified tag name.
- **querySelector(selector):** Returns the first element that matches a specified CSS selector.
- **querySelectorAll(selector):** Returns a NodeList of all elements that match a specified CSS selector.

Matches, closest & contains method

There are three important methods to search the DOM

1. **element.matches(css) :** To check if element matches the given css selector
2. **element.closest(css):** To look for the nearest ancestor that matches the given css selector. The element itself is also checked.
3. **Element.contains(element):** return true if element B inside elementA or when elementA == elementB

DOM Events

Event: An action performed by the user on the webpage is known as an event

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

- Click
- Mouseover
- Mouseout
- Mousedown
- Mouseup
- Doubleclick
- Keypress
- Keyup
- Keydown

DOM Event Listener

- The addEventListener() method attaches an event handler to the specified element
- The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)
- The second parameter is the function we want to call when the event occurs.

Ex:

```
element.addEventListener("click", function () {
    alert("Hello World!");
});
```

Event Propagation

Event propagation refers to the way events travel through the DOM tree. When an event occurs on an element, it can trigger event handlers not only on that element but also on its parent elements, all the way up to the root of the document.

There are two phases of event propagation:

Capturing Phase: The event travels from the root of the DOM tree down to the target element.

Bubbling Phase: The event then bubbles up from the target element back to the root.

event.stopPropagation(): This method prevents further propagation of the current event. It stops the event from bubbling up the DOM tree or from capturing down the tree

stopPropagation() in bubbling phase, it will block the progation to reach to the outer moat element from the targeted element

stopPropagation() in capturing phase, it will block the progation to reach to the targeted element

JSON(Java Script Object Notation)

- It is format of storing and transporting data.
- It is used when the data is sent form the server to web page
- It is a popular data format for developers because of its human readable text, light weight which requires less coding and faster process

Example:

```
[
     {"fname":"Ram","lname":"Patil"},
     {"fname":"Raj","lname":"smith"},
     {"fname":"Ashish","lname":"gouda"}
 ]
```

Syntax of JSON

- Data is in Key value pairs
- Data is separated by ','
- Curly braces hold objects
- [ ]---holds array

In JavaScript, there are two main methods used to work with JSON (JavaScript Object Notation):

1. **JSON.Stringify()** : used to convert normal object to JSON format
2. **JSON.parse():** Used to convert JSON format of data to normal object

Time delays

setTimeOut() and setInterval() are two functions in js allows to schedule tasks to be executed at later time. Used for animations, polling data from server and other asynchronous tasks.

setTimeOut() :

- Runs the code with time delay

Syntax :

```
window.setTimeOut(function , delay);
```

function: The function you want to execute.

- **delay:** The amount of time (in milliseconds) before the function is executed

setInterval() :

The function similar to setTimeOut, but it schedules a function to be executed repeatedly at specified interval.

Syntax :

```
window.setInterval(function , delay);
```

function: The function you want to execute.

delay: The amount of time (in milliseconds) before the function is executed

- createTimeOut() and createInterval()
- cancel a setTimeout, you use clearTimeout
- cancel a setTimeout, you use clearTimeout

Promises

In JavaScript, promises are a mechanism for handling asynchronous operations. They provide a way to work with asynchronous code in a more structured and readable manner. Promises represent a value that may not be available yet but will be resolved at some point in the future, either successfully with a result or with an error.

A promise represents the eventual result of an asynchronous operation.

It can be in one of three states:

- **Pending:** Initial state, neither fulfilled nor rejected.
- **Fulfilled:** Meaning the operation completed successfully.
- **Rejected:** Meaning the operation failed.

Syntax to create promise

```
const myPromise = new Promise((resolve, reject) => {
    // Asynchronous operation, e.g., fetching data from a server.

    if (/* operation successful */) {
        resolve('Success data'); // Resolve if successful.
    } else {
        reject('Error message'); // Reject if there's an error.
    }
});
```

Consuming a Promise:

- **Then() :** It will execute when the promise will be in resolved state
- **Catch() :** It will get executed when the promise is in rejected state
- **Finally() :** It will execute always means promise is in resolve , reject or in pending state

Promise methods

1. **Promise.all()** all promise should be in resolve state
2. **Promise.allSettled()** either resolve/ reject, then() block will execute or else catch() block..[catch block output we can't see]
3. **Promise.any()** at least 1 or more promises should be resolved, then() will execute or else catch() block.
4. **Promise.race()** it depends upon time, whichever comes first that promise will get executed.

Fetch API

- fetch() is used to send a request to backend, it accepts a URL/API as an argument. This url should be in String
- The fetch method returns a promise object, This object needs to be handled using then() and catch()
- We need to first handle the response object and parse it to get normal object using json()
- In the next then method we will be able to consume the data (using data from backed is called consuming data)

EXAMPLE:-

```
        fetch("https://api.github.com/users")
      .then((response) => response.json())
      .then((data) => {
          console.log(data[0].login);
      })
      .catch((error) => {
          console.log(error);
      });
```

Async and await

- *"async and await make promises easier to write"*
- async makes a function return a Promise
- await makes a function wait for a Promise

Async : The async keyword is used to define a function that returns a promise. It allows you to write

Await : The await keyword is used inside an async function to wait for a Promise to settle (either resolve or reject). It can only be used inside an async function

EXAMPLE:-

```
  let tbody = document.getElementById("tbody");

   let fetchData = async function () {
       let response = await fetch("https://api.github.com/users");
       let data = await response.json();
       displayData(data);
   };
   fetchData();
   let str = "";
   function displayData(apiData) {
       apiData.forEach((ele) => {
           str += `
<tr>
```

```
<td>${ele.login}</td>
<td>${ele.id}</td>
<td><a href='${ele.url}' class = 'link'>link</a></td>
<td><img src='${ele.avatar_url}'></td>
</tr>
`;
            tbody.innerHTML = str;
        });
    }
```

Object Oriented programming

In programming we often take something and then extend it. For example, we might create a user object and “admin” and “guest” will be slightly modified variants of it.

Prototype

JavaScript objects have special property called prototype that is either null o =r reference of another object.

When we try to read a property from a prototype and it is missing then JavaScript automatically takes it from the prototype. This is called “prototype inheritance”.

Setting prototype

We can set prototype by setting __proto__.

Now if we read a property from the object which is not in object and is present in the prototype, JavaScript will take it from prototype.

If we have a method in object, it will be called from the object. If it is missing in object and present in prototype, it called from the prototype.

Classes and Objects

In Object oriented programming, a class is an extensible program- code template for creating, providing initial values for state(member variable) and implementation of behavior(member functions).

The basic syntax of writing a class is

```
Class MyClass{
        // class methods
        Constructor(){.....}
        Method1(){.....}
        Method2(){.....}
}
```

We can the use new MyClass() to create a new object with all the listed methods.

The constructor method

The constructor method is automatically called by new keyword, so we can initialize the object there.

Class Inheritance

Class inheritance is the way for one class to extend another class. This way is done using the **extends** keyword.

The extends keyword

Extends keywords are used to extend another class.

Method Overriding

If we create our own implementation of run, it will not be taken from the Animal class. This is called method overriding.

Super Keyword

When we override a method, we don't want the method of the previous class to go in vain. We can execute it using super keyword.

Overriding constructor

With a constructor, things are a bit tricky/different according to the specification, if a class extends another class and has no constructor, then the following empty constructor is generated.

Constructor in inhering class must call super() and do it before using this().

We can also use super() method in a child method to call parent method.

Static Method

Static methods are used to implement functions that belongs to a class as a whole and not to only particular object.

Objects in JS

- Object is nothing but the thing which has existence in the world.
- A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- In java script object is used to store the data in key value pairs.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- Objects are mutable.
- We can access , update, add and delete the properties from an object.
- A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- Creating Objects in JavaScript:
- **There are 3 ways to create objects.**

1. By object literal.
2. By creating instance of Object directly (using new keyword).

1. **JavaScript Object by object literal:**

The syntax of creating object using object literal is given below:

```
VariableName object =
    {
        property1: value1,
        property2: value2,
        propertyN: valueN
    }
```

As you can see, property and value is separated by : (colon).

Example of creating object in JavaScript.

```
    let emp = {
        id: 102,
        name: "Kumar", salary: 40000
    }
    document.write(emp.id + " " + emp.name + " " + emp.salary);
```

2. **By creating instance of Object:**

The syntax of creating object directly is given below:

```
    var objectname = new Object();
```

Here, **new keyword** is used to create object.

Example of creating object directly.

```
    var emp = new Object(); emp.id = 101; emp.name = "Ravi"; emp.salary = 50000;
    document.write(emp.id + " " + emp.name + " " + emp.salary);
```

In JavaScript, to access and manipulate object properties: dot notation

Dot notation

Dot notation is the most common way to access object properties. It uses a period (.) to access the value of a property by its key.

Here's an example:

```
    const
        person = {
            name: 'John', age: 30, address: {
                street: '123 Main St',
                city: 'New York'
```

```
        }
    };
console.log(person.name); // John  console.log(person.address.city); // New York
```

OBJECT METHODS

- *Objects can also have methods.*
- *Methods are actions that can be performed on objects.*

OBJECT METHODS:

1.keys : *It will return array of keys*

2.Values : *It will return array of values*

3.Entries : *It will return array of keys and values*

4.Assign : *It is used to merge two objects*

5.Seal : *We can only update the properties*

6.isSealed : *It is used to check whether the particular object is sealed or not*

7.Freeze : *We cannot do any modifications in an object*

8.Isfrozen : *It is used to check whether the particular object is frozen or not*

```
let obj4 = {
    ename: "lavanya",
    id: 123,
    sal: 20000
};
let obj5 = {
    ename1: "bujji", id1: 12, sal1: 40000
};
console.log(Object.keys(obj4));//array of keys
console.log(Object.values(obj4))//array of values
console.log(Object.entries(obj4))//array of keys and values
console.log(Object.assign({}, obj4, obj5))//merge multiple objects
console.log(obj4)//stroring all properites of obj5 into obj4
Object.seal(obj4);//In seal method i can update can't add,i can't remove,can
fetch
console.log(Object.isSealed(obj4));//true

delete obj4.ename; console.log(obj4)
obj4.sal = 12000; console.log(obj4); console.log(obj4.sal);
Object.freeze(obj4); console.log(Object.isFrozen(obj4))//true  obj4.skills="html";
obj4.sal = 300; console.log(obj4)
console.log(obj4)//freeze object we can't add, we can't update, we can't  delete, we can
fetch
delete obj4.id
console.log(obj4);
```

LOOPS IN JAVASCRIPT

- Loops can execute a block of code a number of times.
    - **forEach** - loops through a block of code a number of times
    - **for/in -** loops through the properties of an object
    - **for/of -** loops through the values of an Iterable object

1. forEach():-

The forEach() method calls a function for each element in an array.

The forEach() method is not executed for empty elements.

Example:

```
const array1 = ['a', 'b', 'c’]
    array1.forEach((element) => console.log(element));

    Output:-  1
2
3
4
5
```

2. For of loop:-

Iterable objects are objects that can be iterated over with for..of. <script> Ex:-const name = "W3Schools";

```
    let text = ""
    for (const x of name) {
        text += x + "<br>";
    }
```

3. For in loop:-

The JavaScript for in loop iterates over the keys of an object

Ex:-

```
    const object = { a: 1, b: 2, c: 3 }; for (const property in object) {
        console.log(`${property}: ${object[property]}`);
    }

    // Expected output:
    // "a: 1“
    // "b: 2“
    // "c: 3“
```