INDEX

JAVASCRIPT

INTRODUCTION:-

It is a light-weight object-oriented programming language which is used by several websites for scripting WebPages. It is an interpreted full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. JavaScript was invented by Brendan Eich in 1995. It is client side scripting language.

FEATURES:-

- All popular web browsers support javascript as they provide built-in execution environments
- Javascript follows the syntax & structure of C programming language.
- Javascript is light-weight & interpreted language as known scripting language.
- Javascript is case-sensitive and dynamic type language.
- Javascript is supportable in several operating systems including windows, mac OS etc.
- It provides good control to the users over the web browsers.

IMPLEMENTATION:-

Javascript can implement in "script" tag of html in head or body tag it is called in-page javascript also external javascript file can link using "src" attribute in script tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
    <script>
    …
    </script>
   <script type="text/javascript" src="file_name.js"></script>
  </head>
  <body>
    <script>
      …
    </script>
  </body>
</html>
```

Example:-

```
<script>
     Document.write("hello world");
</script>
```

HTML TAGS IN JAVASCRIPT:-

```
Document.write("hello <br>");
document.write("<font color='red'>JavaScript</font>");
```

LANGUAGE BASICS

COMMENTS:-

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments

Single line comment:-

It is represented by double forward slashes (//). It can be used before and after the statement.

Example:-

```
// here is comment
```

Multi-line comment:-

It can be used to add single as well as multi line comments. So, it is more convenient. It is represented by forward slash with asterisk then asterisk with forward slash.

Example:-

```
/* here is multi-
Line comment */
```

VARIABLES:-

A JavaScript variable is simply a name of storage location.

Rules:-

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Var:- declare same variable name multiple time with assign different values. Also it is global scope variable.

```
var x=90;
```

let:- can declare variable once assign value multiple time. It is block scope variable.

```
Let v=9;
```

Const(constant):- cannot declare same name multiple time or assign value to it.

```
Const b=45;
```

Print variable with ${} and backtick:-

Example: - var a=5; document.write(`value of a=${a}`);

SCOPE OF VARIABLE:-

Global variable:-

Variables declare outside of block are global variable also it is declare with **var** it can access within block also. Also with window object we can declare global variable inside block access them from other block.

Example:-

```
Window.num=34;
Var a=89;
```

Local variable:-

Variables that declared and access inside block or function is local variables. **Let** is used for declare local scope variable.

Example:-

```
    Let a=78;
```

STRICT MODE:-

Strict mode is declared by adding "use strict"; to the beginning of a script. Strict mode makes it easier to write "secure" JavaScript.

As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

Example:-

```
“use strict”;
X=89;//this will throw error that x is not declared.
```

HOISTING:-

Hoisting is a mechanism in JavaScript that moves the declaration of variables and functions at the top. So, in JavaScript we can use variables and functions before declaring them

Example:-

```
X=34;
Doucument.write(x);
Var x;
```

DATA TYPES:-

JavaScript is a dynamic type language, means you don't need to specify type of the variable.

String: - represents sequence of characters. Also concatenation of string is possible using “+” operator.

Example: - var a=”hello”;

Number: - represents numeric values.

Example: - var a=45;

Boolean: - represents Boolean value either false or true

Example: - var a=true;

Undefined: - represents undefined value

Example: - var a;

Null: - represents null i.e. no value at all

Example: - var a=null;

Typeof: - this operator is used for check type of value in javascript **example**: - typeof a

CONSOLE:-

Console is available in developer tool. Press F12 to open console.

Console.log("hello");	print on console.
Console.error("error");	show error massage.
Console.warn("warning");	show warning.
Console.clear();	clearing console.

BOX:-

Alert box:-

It will show message in alert box.

Example:-

```
Alert("message");
```

Confirm box:-

It will show message with "ok" and "cancel" for confirmation. Value of "ok" is 1 and value of "cancel" is 0.

Example:-

```
Confirm("do you like it?");
```

Prompt box:-

It is use to take user input and only take string type.

Example:- Prompt("enter name");

DEBUGGING:

Sometimes a code may contain certain mistakes. Being a scripting language, JavaScript didn't show any error message in a browser. But these mistakes can affect the output.

Console.log():-

The console.log() method displays the result in the console of the browser. If there is any mistake in the code, it generates the error message.

Example:-

```
X=10;
Console.log(a);
```

Debugger keyword:-

In debugging, generally we set breakpoints to examine each line of code step by step. There is no requirement to perform this task manually in JavaScript. JavaScript provides **debugger** keyword to set the breakpoint through the code itself. The debugger stops the execution of the program at the position it is applied.

Example:-

```
X=10;
Document.write(x);
Debugger;
Document.write(a);
```

OPERATORS:-

Arithmetic operator:-

Operator	**Description**	**Example**
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Example:-
Var=Number(34) convert value to number
Var a=34,b=45,c=a+b;
Document.write(c);

Comparison operator:-

Operator	**Description**	**Example**
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Example:-
Var a=9,b=10,c=a>b;
Document.write(c);

Logical operator:-

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Assignment operator:-

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Example:-
Var a=98;a+=2;
Document.write(a);

CONTROL STATEMENTS

IF STATEMENT:-

It evaluates the content only if condition is true.

Syntax:-

```
If(condition){
        Statements.
}
```

Example:-

```
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
```

IF ELSE STATEMENT:-

It evaluates the content whether condition is true of false.

Syntax:-

```
If(condition){
        Statements.
}
else {
        Else’s statements.
}
```

Example:-

```
var a=50;
if(a>20){
document.write("a is greater than 20");
}
else{
document.write(" a is smaller than 20");
}
```

CONDITIONAL OPERATOR (?:):-

Syntax:-

```
        Expression1?expression2:exprssion3
```

Example:-

```
        Var a=8,b=6;
        a>b?document.write(“a is greater”): document.write(“b is greater”);
```

ELSE IF LADDER STATEMENT:-

It evaluates the content only if expression is true from several expressions.

Syntax:-

```
If(condition1){
        Statements.
}
… … …
Else if(condition N) {
        Statements.
}
else {
        Else's statements.
}
```

SWITCH STATEMENT:-

The JavaScript switch statement is used to execute one code from multiple expressions. It is just like else if statement

Syntax:-

```
Switch(expression)
{
   Case value1:
        Statements;
        Break;
   … … …
   Case valueN:
        Statements;
        Break;
   Default:
        Statements;
}
```

Example:-

```
let a=7;
switch(a){
    case 1:
        document.write("rank "+a);
        break;
    case 2:
        document.write("rank "+a);
        break;
    default:
        document.write("unranked");
}
```

LOOPS

WHILE LOOP:-

It is an **entry control loop**. That checks condition before entering loop and iterate until condition remain true.

Syntax:-

```
Initialization
While(condition)
{
    Statements;
}
```

Example:-

```
Let a=1;
While(a<10)
{
        document.write("hello ",a,"<br>");
          a++;
}
```

DO WHILE LOOP:-

It is an **exit control loop**. It execute at least one time whether condition is false or true.

Syntax:-

```
Do{
     Statements;
}while(condition);
```

Example:-

```
Let a=1;
Do{
     Document.write(a,"<br>");
}while(a<=10);
```

FOR LOOP:-

It is loop with all initialization, condition and increment/decrement in single line, separated by semicolon [;].

Syntax:-

```
For(initialization;condition;increment/decrement)
{
 Statements;
}
```

Example:-

```
for(i=0;i<10;i++)
{
      document.write("hello "+i+"<br>");
}
```

BREAK AND CONTINUE:-

Break: - it is keyword used for break loop.

```
for(let i=0;i<10;i++)
{
  if(i==5)
  {
    break;
  }
  document.write("looping<br>");
}
```

Continue: - it skips remaining code after execution of continue and start next iteration of loop.

```
for(let i=0;i<10;i++)
{
  if(i<=5)
  {
    continue;
  }
  document.write(i,"looping<br>");
}
```

FUNCTIONS

ADVANTAGE:-

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

1) **Code reusability:** We can call a function several times so it saves coding.
2) **Less coding:** It makes our program compact. We don't need to write many lines of code each time to perform a common task.

FUNCTION SYNTAX:-

Use **function keyword** to define function.

```
Function function1(){
        Function's statements;
}
```

Example:-

```
Function hello(){
        Document.write("hello from function");
}
Hello();
```

FUNCTION ARGUMENT:-

Functions can take arguments if it needs to operate with outside values.

Syntax:-

```
Function func1(parameter1,… …,parameterN){
        Statements;
}
```

Example:-

```
Function square(num){
        Document.write(num*num);
}
```

FUNCTION RETURN VALUE:-

Function also returns value to store in outside variable.

Syntax:-

```
Function func1(parameter1,… …,parameterN){
        Statements;
        Return value;
}
```

Example:-

```
Function sum(num1,num2){
        Return num1+num2;
}
```

ARRAYS

JavaScript array is an object that represents a collection of similar type of elements. It is homogeneous data structure. It takes index from 0.

DECLARE AND ACCESS ARRAY:- ARRAY LITERAL

Syntax:-

```
Var arrayname = [value1, value2, … …, valueN];
Arrayname[index];
```

Example:-

```
Var student = [“ram”,”shyam”,”vishnu”];
Document.write(student[1]);
```

EMPTY ARRAY:- ARRAY DIRECTLY

Here, **new keyword** is used to create instance of array.

Syntax:-

```
Var arrayname = new Array();
```

Example:-

```
var student = new Array();
emp[0]=”shyam”;
emp[1]=”ram”;
emp[2]=”hari”;
```

FOR/OF LOOP: - for iterable array

```
for(let Key of array1)
{
        console.log(key)
}
```

FOREACH LOOP:-

First parameter of function in foreach loop is value, and second is index.

```
a.forEach(loop);
function loop(value,index){
        document.write(index+":"+value+"<br>");
}
```

ARRAY METHOD:-

Sr.	Methods	Description
1	Array1.sort();	Sort array lexicographically.
2	Array1.reverse();	Reverse array element.
3	Array1.pop();	Delete last element of array.
4	Array1.push(elements);	Add values at last position.
5	Array1.shift();	Delete first element.
6	Array1.unshift(elements);	Add values at first position.
7	ArrayN=array1.concat(array2);	Add two or more array in new array.
8	Variable1=array1.join(" ");	Convert array element in single string join with argument.
9	arrayN=array1.slice(from, to);	Slice elements from array and add them to new one. Use negative index also last index is equal to -1.
10	Array1.splice(pos, remove element amount, new values);	Add new elements at position in array and remove existing elements from that position if need.
11	Variable1=array1.indexOf(element);	Return index of element if not found return -1.
12	Variable1=array1.includes(element);	Return true if match element value and datatype.
13	Array1.fill(value);	Replace all array elements by value.
14	Variable1=Array1.length;{property}	Show array length or total element count.
15	Variable1=array1.toString();	Convert array into string.
16	Variable1=array1.map(func)	Return new array after doing operation by function

Example: - (map)

```
arr=[2,4,6,8,10]
newarr= arr.map(add)
document.write(arr,"<br>")
document.write(newarr,"<br>")

function add(num)
{
  return num*10
}
```

OBJECTS

A JavaScript object is an entity having state and behavior (properties and method).

CREATE OBJECT:- OBJECT LITERAL

Syntax:-

```
Object1={property1:value1,… …, propertyN:valueN};

Document.write(object1.property1);
```

Example:-

```
var a={
   name:"ram",
   age:78,
   "last":"suryawanshi"
};
document.write(a["last"]);
document.write(a.name);
```

EMPTY OBJECT:- INSTANCE OF OBJECT

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

Syntax:-

```
Object1= new Object();
Object1.property=value;
```

Example:-

```
var a= new Object();
a["last name"]="raghuwanshi";
a.name="ram";
```

FOR/IN LOOP:-

```
for(let Key in a)
{
        document.write(Key+":"+a[Key]+"<br>");
}
```

OBJECT METHOD:-

Sr.	Method	Description
1	Object.defineProperty(obj1,‘Property1’,{value:”xyz”});	Method is used for define property to object.
2	Object.defineProperties(obj1,{ Property1:{value:”abc”}, Property2:{value:”xyz”} });	Method is used for define multiple properties to object.
3	Var1=Object.entries((obj1));	Method return property and value of given property of object as array.
4	Var1=Object.values(obj1);	Return values of object as array.
5	Var1=Object.keys(obj1);	Return keys of object as array

Example:-

```
obj={name:"good",age:34,marks:567}

console.log(obj)
Object.defineProperties(obj,{
   name:{value:"food"},
   age:{value:20},
   score:{value:45}
});
console.log(obj)
```

STRING

CREATE STRING:-

The JavaScript string is an object that represents a sequence of characters.

String literal: -

Syntax:-

Var1="string value";

Example:-

Let good="good day";

String object: - new keyword is used to create instance of string.

Syntax:-

Var1=new String("string value");

Example:-

Let hello= new String("hello good day");

For loop:-

```
let a = "string power";
for(key in a){
  document.write(a[key],"<br>");
}
```

STRING METHOD:-

Sr.	Method	Description
1	Var1=str1.charAt(index);	Return character is store at given index.
2	Var1=str1.charCodeAt(index);	Return ASCII code of character store at given index
3	Var1=str1.concat(str2);	Return two joined string.
4	Var1=str1.indexOf('char');	Return index of given character.
5	Var1=str1.lastIndexOf('char');	Return index of given character search from last position.
6	Var1=str1.search('string');	Return index if match found else -1.
7	Var1=str1.match('string');	Return string if match found else null.
8	Var1=str1.replace('oldstr','newstr');	Replace old string with new one.
9	Var1=str1.substr(index,length);	Return character in string from given index and length.
10	Var1=str1.substring(start index,end index);	Return characters from start to end index default end of string.
11	Var1=str1.slice(start index,end index);	Return character from start to end index default end of string also take negative index.
12	Var1=str1.toLowerCase();	Convert string to lower case.
13	Var1=str1.toUpperCase();	Convert string to upper case.
14	Var1=str1.split(" ");	Split from given argument return new array of it.
15	Var1=str1.trim();console.log(var1);	Remove white spaces from start and end of string.

NUMBER

The JavaScript number object enables you to represent a numeric value. It may be integer or floating-point.

SYNTAX:-

```
Var n = new Number(value);
```

Create number with number constructor in javascript using new keyword. If value cannot convert in number it returns NaN. (NaN = Not a Number).

Example:-

```
Var n = new Number(45);      //integer value by number object.
Var n = 234;                 //integer value.
Var n = 12.34;               //floating point value.
Var n = 12e2;                //exponent value. Output: 1200.
```

NUMBER CONSTANT:-

Sr.	Constant	Descriptions
1	Var1 = Number.MIN_VALUE;	Return minimum value in javascript.
2	Var1 = Number.MAX_VALUE;	Return maximum value in javascript.
3	Var1 = Number.POSITIVE_INFINITY;	Return positive infinity, overflow value.
4	Var1 = Number.NEGATIVE_INFINITY;	Return negative infinity, overflow value.
5	Var1 = Number(“hello”); output:NaN;	Return NaN not a number.

NUMBER METHOD:-

Sr.	Method	Descriptions
1	Var1 = Number.isFinite(num); 10/0	Return true if given value is finite.
2	Var1 = Number.isInteger(num);	Return true if given value is integer.
3	Var1 = Number.isNaN(num);	Return true if given value is NaN.
4	Var1 = Number.parseFloat(“string”);	Convert string in floating number.
5	Var1 = Number.parseInt(“string”);	Convert string in integer number.
6	Var1 = num.toExponential();	Return number in exponential form.
7	Var1 = num.toFixed(); 12.34 -> 12	Return number in round (fixed) format.
8	Var1 = num.toPrecision(digit); 12.37 (3)->12.4	Return precise number in given digit.
9	Var1 = num.toString();	Return number into string format.

MATH

The JavaScript math object provides several constants and methods to perform mathematical operation

MATH METHOD:-

Sr.	Method	Descriptions
1	Var1=Math.abs(num); -45 -> 45	Return absolute value of number.
2	Var1=Math.cbrt(num);	Return cube root of given number.
3	Var1=Math.ceil(num); 34.12 -> 35	Return largest integer value of given number.
4	Var1=Math.floor(num); 34.98 -> 34	Return smallest integer value of given number.
5	Var1=Math.hypot(num1,num2);	Return hypotenuse of given numbers.
6	Var1=Math.max(num1, …,numN);	Return maximum value of given numbers.
7	Var1=Math.min(num1, …,numN);	Return minimum value of given numbers.
8	Var1=Math.pow(base,exponent);	Return power of given number.
9	Var1=Math.random();	Return random number betweens 0 to 1.
10	Var1=Math.round(num);	Return nearest integer value of given number.
11	Var1=Math.sign(num); -1,0,1	Return sign of given number.
12	Var1=Math.sqrt(num);	Return square root of given number.
13	Var1=Math.trunc(num);	Return integer value of given number.
14	Var1=Math.PI;(property)	Return PI value.

DATE METHOD:-

Sr.	Method	Descriptions
1	date= **new** Date();	Return date as day, month, date, year, time, time zone.
2	Var1=date.getDate();	Return date as 1 to 31.
3	Var1=date.getDay();	Return day as 0 to 6. 0=Sunday.
4	Var1=date.getFullYear();	Return year.
5	Var1=date.getHours();	Return 0 to 23 represents hour.
6	Var1=date. getMilliseconds();	Return milliseconds as 0 to999.
7	Var1=date. getMinutes();	Return minutes as 0 to 59.
8	Var1=date.getMonth();	Return month as 0 to 11.
9	Var1=date.getSeconds();	Return seconds as 0 to 59.
10	Var1=date.toDateString();	Return date portion in string.
11	Var1=date.toISOString();	Return date in string as ISO format.
12	Var1=date.toTimeString();	Return time in string.

ERRORS

At time of coding violating rules of programming language is occurred error in program.

TYPES:-

1. **Syntax Error:** When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. **Runtime Error:** When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.
3. **Logical Error:** An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

ERROR OBJECT:-

When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. **name:** This is an object property that sets or returns an error name.
2. **message:** This property returns an error message in the string form.

Standard built-in error type:-

1. **EvalError**: It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.
2. **InternalError:** It creates an instance when the js engine throws an internal error.
3. **RangeError:** It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.
4. **ReferenceError:** It creates an instance for the error that occurs when an invalid reference is de-referenced.
5. **SyntaxError:** An instance is created for the syntax error that may occur while parsing the eval().
6. **TypeError:** When a variable is not a valid type, an instance is created for such an error.
7. **URIError:** An instance is created for the error that occurs when invalid parameters are passed in encodeURI() or decodeURI().

TRY... CATCH STATEMENT:-

try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

catch{} statement: This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.

Syntax:-

```
Try{
        // code to be written.
}
Catch(var1){
        // code for error handling.
}
```

Example:-

```
Try{
        Document.write("good day");
}
Catch(error)
{
        Console.log(error.name);
        Console.log(error. message);
}
```

THROW STATEMENT:-

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

Syntax:-

```
Try{
        Throw exception;        //user define exception.
}
Catch(var1){
        Expression;      // code for handling exception.
}
```

Example:-

```
try{
  a=67;
 if((typeof a)!="string"){
     throw new Error("Not a string");
  }
  else{
     document.write(a);
  }
}
catch(e){
     e.name="string error";
     console.log(e.name);
     console.log(e.message);
}
```

DOM

The DOM stands for Document Object Model. The document object represents the whole html document. When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

PROPERTIES:-

The properties of document object that can be accessed and modified by the document object.

Example:-

Properties	Description
document.all;	Targeting all element.
document.head;	Targeting head element.
document.title;	Targeting title element.
document.body;	Targeting body element.
document.links;	Targeting all anchor tag.
document.images;	Targeting all images.
document.forms;	Targeting forms.
document.doctype;	Return DOCTYPE.
document.URL;	Return url.

DOM CHILD:-

Elements that directly nested in other element is called children of that element.

Child nodes: - accessing the child nodes of any elements includes all nodes.

Example:-

```
Document.body.childNodes;
Document.body.childNodes[1];
```

First and last child: - accessing first and last child of parent element.

```
Document.body.firstChild;
Document.body.lastChild;
```

Children: - access only html tag elements and nodes.

Example:-

```
Document.body.children;
Document.body.children[1];
```

First and last element child: - accessing first and last child element child of parent element.

```
Document.body.firstElementChild;
Document.body.lastElementChild;
```

Parent element of child: - accessing parent element of child.

```
Document.body.parentElement;
```

Sibling elements: - accessing the previous and next element of element.

Example:-

```
Document.body.children[3].previousElementSibling;
Document.body.children[3].nextElementSibling;
```

DOM PROPERTIES:-

innerHTML:- it can be used to write the dynamic html on the html document. It is used for get or set value.

Syntax:-

```
Target.innerHTML;
```

Example:-

```
<div class=contain>
   Good
</div>
<script>
   a=window.document.body;
   b=a.children[0].innerHTML="<h1>day</h1>";
   c=a.children[0].innerHTML;
   console.log(c);
</script>
```

innerText:- it can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

Syntax:-

```
Target.innerText;
```

Example:-

```
<div class=contain>
   Good
</div>
<script>
   a=window.document.body;
   b=a. children[0].innerText="day";
   c=a. children[0].innerText;
   console.log(c);
</script>
```

Style: - it can used to change CSS properties of elements.

Syntax:-

```
Target.style.property="value";
```

Example:-

```
    <div class=contain>
       Good
    </div>
    <script>
       a=window.document.body;
       b=a.children[0].style.backgroungColor=”red”;
        c=a.children[0].style.backgroungColor;
       console.log(c);
    </script>
```

DOM METODS:-

GetElementById(): - returns the element of specified id.

Syntax:-

```
        Document.getElementById(“ID”);
```

Example:-

```
    <div id=contain>
       Good
    </div>
    <script>
       a=window.document;
       b=a.getElementById("contain");
       console.log(b);
    </script>
```

getElementsByClassName(): - The getElementsByClassName() method is used for selecting or getting the elements through their class name value. On calling the getElementsByClassName() method on any particular element, it will search the whole document and will return only those elements which match the specified or given class name.

Syntax:-

```
        Document.getElementsByClassName(“Class”);
```

Example:-

```
    <div class=contain>
       Good
    </div>
    <script>
       a=window.document;
       b=a.getElementsByClassName("contain")[0];
       console.log(b);
    </script>
```

getElementsByName(): - returns all the element of specified name.

Syntax:-

```
    Document.getElementsByName(“Name”);
```
Example:-
```
  <div class=contain>
    Good
  </div>
  <script>
    a=window.document;
    b=a.getElementsByName("Name")[0];
    console.log(b);
  </script>
```

getElementsByTagName(): - returns all the element of specified tag name.

Syntax:-
```
    Document.getElementsByTagName(“Tag”);
```
Example:-
```
  <div class=contain>
    Good
  </div>
  <script>
    a=window.document;
    b=a.getElementsByTagName("div")[0];
    console.log(b);
  </script>
```

querySelector(): - return first element from document that match to argument.

Sr.	Selector	Access
1	Tag	TagName
2	Class	.className
3	Id	#idName

Syntax:-
```
    Document.querySelector(“element”);
```
Example:-
```
  <div class=contain>
    Good
  </div>
  <script>
    a=window.document;
    b=a. querySelector ("div");
    console.log(b);
  </script>
```

querySelectorAll(): - return all elements from document that match to argument.

Syntax:-

```
 Document.querySelectorAll(“element”);
```
Example:-
```
<div class=contain>
  Good
</div>
<script>
  a=window.document;
  b=a. querySelector All(".contain");
  console.log(b);
</script>
```

DOM ATTRIBUTE:-

attributes:- to see all attribute of html tags.
Syntax:- target.attributes;
Example:-
```
let box=document.getElementById("box");
b=box.attributes;
console.log(b);
```

getAttribute:- to get value of attribute.
Syntax:- target.getAttribute(‘attribute_name’);
Example:-
```
let box=document.getElementById("box");
let a=box.getAttribute('class');
console.log(a);
```

setAttribute:- to set value of attribute.
Syntax:- target.getAttribute(‘attribute_name’,’attribute_value’);
Example:-
```
let box=document.getElementById("box");
box.setAttribute('class','gd');
```

removeAttribute:- to remove attribute.
Syntax:- target.removeAttribute(‘attribute_name’);
Example:-
```
let box=document.getElementById("box");
box.removeAttribute('class');
```

CREATE & INSERT ELEMENTS:-

createElement:- to create an element(tag).
Syntax:- document.createElement(‘element_name’);
Example:-
```
let b=document.createElement('div')
b.setAttribute('class','gd');
```

append:- to insert an element(tag) in end of target.
Syntax:- target.append('element');
Example:-

```
        let b=document.createElement('div')
        document.getElementById('good').append(b);
```

prepend:- to insert an element(tag) in start of target.
Syntax:- target.prepend('element');
Example:-

```
        let b=document.createElement('div')
        document.getElementById('good').prepend(b);
```

before:- to insert an element(tag) before target.
Syntax:- target.before('element');
Example:-

```
        let b=document.createElement('div')
        document.getElementById('good').before(b);
```

after:- to insert an element(tag) after target.
Syntax:- target.after('element');
Example:-

```
        let b=document.createElement('div')
        document.getElementById('good').after(b);
```

remove:- to remove an element(tag).
Syntax:- target.remove();
Example:-

```
        document.getElementById('good').remove();
```

DOM CLASS:-

className:- return string value of class.
Syntax:- target.className;
Example:-

```
        a=document.getElementById("contain").className;
        console.log(a);
        document.getElementById("contain").className='c2';        //assign class name
```

classList:- return array of class names.
Syntax:- target.classList;
Example:-

```
        a=document.getElementById("contain").classList;
        console.log(a);
```

add class:- add class in class list.
Syntax:- target.classList.add('class_name');
Example:-

```
document.getElementById("contain").classList.add(‘c4’);
```

remove class:- remove class in class list.
Syntax:- target.classList.remove(‘class_name’);
Example:-

```
document.getElementById("contain").classList.remove(‘c1’);
```

toggle class:- toggle class in class list.
Syntax:- target.classList.toggle(‘class_name’);
Example:-

```
document.getElementById("contain").classList.toggle(‘c3’);
```

BOM

The BOM stands for Browser Object Model. It is used to interact with the browser. The default object of browser is window means you can call all the functions of window by specifying window or directly.

WINDOW OBJECT:-

It represents a window in browser. An object of window is created automatically by the browser. It is the object of browser; it is not the object of javascript. The javascript objects are string, array, date etc.

METHODS:-

Alert(): - displays the alert box containing message with ok button.
Example:-
```
Alert(“good day”);
```

Confirm(): - displays the confirm dialog box containing message with ok and cancel button.
Example:-
```
Confirm(“are you sure”);
```

Prompt(): - displays a dialog box to get input from the user.
Example:-
```
Prompt(“enter age”);
```

Open():- opens the new window.
Example:-
```
Open(“https://www.google.com”);
```

setTimeout():- performs action after specified time like calling function, evaluating expressions etc.
```
setTimeout(“function”,milliseonds);
```
Example:-
```
setTimeout("hello()",2000);
function hello(){
  document.write("hello world");
}
```

setInterval():- performs same action again and again after some particular time with function.
```
setInterval(“function”,milliseconds);
```
Example:-
```
setInterval("hello()",2000);
function hello(){
  document.write("hello world");
}
```

HISTORY OBJECT:-

The JavaScript history object represents an array of URLs visited by the user.

METHODS:-

History.length:- returns the length of the history URLs.
Example: - console.log(history.length);

History.forward(): - loads the next page.
Example: - console.log(history.forward());

History.back(): - loads the previous page.
Example: - console.log(history.back());

History.go(): - loads the given page number.
Example: - console.log(history.forward(2)); OR console.log(history.forward(-2));

NAVIGATOR OBJECT:-

PROPERTIES:-

Navigator.appVersion:- returns the version.
Example: - Console.log(navigator.appCodeName);

Navigator.language:- returns the language.
Example: - console.log(navigator.language);

Navigator.userAgent: - return user agent.
Example: - console.log(navigator.userAgent);

Navigator.platform: - returns platform.
Example: - console.log(navigator.platform);

Navigator.onLine: - returns true if browser is online.
Example: - console.log(navigator.onLine);

Navigator.javaEnabled(): - (method) returns true if java is enabled.
Example: - console.log(navigator.javaEnabled());

SCREEN OBJECT:-

The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

PROPERTIES:-

Screen.width: - returns the width of the screen.
Example: - console.log(screen.width);

Screen.height: - returns the height of the screen.
Example: - console.log(screen.height);

Screen.availWidth: - returns the available width of the screen.
Example: - console.log(screen.availWidth);

Screen.availHeight: - returns the available height of the screen.
Example: - console.log(screen.availHeight);

EVENTS

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser.

When javascript code is included in HTML, **JavaScript** reacts over these events and allows the execution. This process of reacting over the events is called Event Handling. Thus, javascript handles the HTML events via Event Handlers.

CALLING EVENT:-

Event calling is possible in html or in javascript also. The action on events is defined in javascript with help of function.

HTML: - calling event in html tag.

Syntax:-

```
<div event_handler=”function_name()” >
```

JAVASCRIPT: - calling event in javascript.

Syntax:-

```
Object.event_handler=function_name;
```

WINDOW EVENTS:-

Load: - When the browser finishes the loading of the page this event take place.
Event Handler: - onload
Example:-
1) <body onload=”hello()”>
2) windows.onload=hello;

Unload: - When the visitor leaves the current webpage, the browser unloads it.
Event Handler: - onunload
Example:-
1) <body onunload=”hello()”>
2) windows.onunload=hello;

Resize: - When the visitor resizes the window of the browser.
Event Handler: - onresize
Example:-
1) <body onresize=”hello()”>
2) windows.onresize=hello;

Offline: - when visitor is offline.
Event Handler: - onoffline

Example:-

1) <body onoffline="hello()">
2) windows.onoffline=hello;

MOUSE EVENTS:-

Click: - When mouse click on an element

Event Handler: - onclick

Example: -

1) <div onclick="hello()">
2) target.onclick=hello;

Double Click: - When mouse click on an element

Event Handler: - ondblclick

Example: -

1) <div ondblclick="hello()">
2) target.ondblclick=hello;

Mouseover: - When the cursor of the mouse comes over the element.

Event Handler: - onmouseover

Example: -

1) <div onmouseover="hello()">
2) target.onmouseover=hello;

Mouseout: - When the cursor of the mouse leaves an element.

Event Handler: - onmouseout

Example: -

1) <div onmouseout="hello()">
2) target.onmouseout=hello;

Mousedown: - When the mouse button is pressed over the element.

Event Handler: - onmousedown

Example: -

1) <div onmousedown="hello()">
2) target.onmousedown=hello;

Mouseup: - When the mouse button is released over the element.

Event Handler: - onmouseup

Example: -

1) <div onmouseup="hello()">
2) target.onmouseup=hello;

Mousemove: - When the mouse movement takes place.

Event Handler: - onmousemove

Example: -

1) <div onmousemove="hello()">

2)	target.onmousemove=hello;

EVENT METHOD:-

addEventListener():- The addEventListener() method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred.

Syntax:- element.addEventListener(event,function);

Example:-

```
document.querySelector("#t").addEventListener("click",change);
function change(){
   document.getElementById("t").style.backgroundColor="aqua";
}
```

FORM EVENTS:-

Focus: - When the user focuses on an element.
Event Handler: - onfocus
Example: -
1)	<input onfocus=”hello()”>
2)	target.onfocus=hello;

Blur: - When the focus is away from a form element.
Event Handler: - onfocus
Example: -
1)	<input onblur=”hello()”>
2)	target.onblur=hello;

Change: - When the user modifies or changes the value of a form element.
Event Handler: - onchange
Example: -
1)	<input onchange=”hello()”>
2)	target.onchange=hello;

Submit: - When the user submits the form. Event works only on submit.
Event Handler: - onsubmit
Example: -
1)	<form onsubmit=”hello()”>
2)	target(form).onsubmit=hello;

Reset: - When the user resets the form. Event works only on submit.
Event Handler: - onreset
Example: -
1)	<form onreset=”hello()”>

2) target(form).onreset=hello;

KEYBOARD EVENTS: - Keyboard events most of works on input tag and body tag only.

Keypress: - event work when any character key is pressed. It returns key value of which character is pressed on keyboard.

Example:-

```
         document.querySelector("body").addEventListener("keypress",h);
        function h(){
           document.getElementById("v").style.backgroundColor="yellow";
        }
```

Keydown: - event work when any key is down. It returns key value of which key is down on keyboard.

Example:-

```
         document.querySelector("body").addEventListener("keydown",h);
        function h(){
           document.getElementById("v").style.backgroundColor="yellow";
        }
```

Keyup: - event work when any key is up. It returns key value of which key is up on keyboard.

Example:-

```
         document.querySelector("body").addEventListener("keyup",h);
        function h(){
           document.getElementById("v").style.backgroundColor="yellow";
        }
```

Action on particular key: - for action on particular key value. Event object will required for take key value. Event object will get as argument in function.

Example:-

```
        document.querySelector("body").addEventListener("keydown",h);
        let e= new Object();     // sometimes it need to declare object.
        function h(e){
           if(e.key=='c')
           {
              document.getElementById("v").style.backgroundColor="yellow";
           }
        }
```

Example: - assign event in body tag with event argument

Html: - <body onkeydown="inpval(event)"> … </body>

Script: -

```
function inpval(e){
   console.log(e)
}
```