FULLSTACK JAVASCRIPT

INTRODUCTION: - information about JavaScript

INFO: -

It is a light-weight, cross-platform 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. It is well-known for the development of web pages, and many non-browser environments also use it. JavaScript was invented by Brendan Eich in 1995. It is client side and server 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>");
```

Semicolon: - use semicolon after complete statement, it will not required if write new statement in new line. JavaScript automatically put semicolon before new line of code.

LANGUAGE BASICS: - basic structure of JavaScript

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 */
```

KEYWORDS:-

These are reserved words in JavaScript those cannot be used as identifiers for variables, functions, classes and objects etc

Identifiers: - these are used to links values with names

Keywords: -

break	Case	Catch	class	Const	continue	debugger	default
delete	Do	Else	export	Extends	false	finally	for
function	If	import	in	instanceof	new	null	return
super	Switch	This	throw	True	try	typeof	var
Void	While	With	let	Static	yield	await	

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}`);

Case-sensitive: - JavaScript identifiers are case sensitive; they identify uppercase and lowercase characters differently.
Ex: -

```
let name, Name;
name="ram";
Name="shyam";
console.log(`name:-${name}, Name:- ${Name}`);
```

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

Literals: - fixed (constants) values that applied to variables and constants are literals

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”);

ESCAPE SEQUENCES: -

These are special characters used for represent characters that cannot be used by using normally using the keyboards

Character	Uses	Character	Uses
\0	Null byte	\'	Single quote
\n	New line	\"	Double quote
\t	Tab	\\	Backslash

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: - JavaScript operators are used to perform different types of mathematical and logical computations (operations). An expression is contains with operators and operand.

Ex: - c=a + b //expression

Operands: - these are values that will get perform in operation

Operators: - these are sign used for perform operations on operands

Ex: - c=a + b // a, b, c: - operands, +, =: - operators

Types of operators in JavaScript: -

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. String Operators
7. Type Operators

Arithmetic operators: - Arithmetic Operators are used to perform arithmetic operations on numbers

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
**	Exponentiation	6**3 = 216
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Example:-

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

Number: - use number method for convert values to number

Ex: -

```
let a=Number(prompt("Enter number1:- "))
```

```
let b=Number(prompt("Enter number2:- "))
let c=a+b;
alert(`Addition of ${a} and ${b} is ${c}`)
```

Exponentiation: - this operator (**) raises the first operand to the power of the second operand.
Ex: -

```
let x = 5;
let z = x ** 2;
console.log(z);
```

Pre and post Increment or decrement: - this operators increment or decrement operand number value by one as a=a+1 or a=a-1, there are pre and post increment or decrement operators
Pre increment or decrement operators first increment or decrement value of variables then used them
Ex: -

```
let a=45;
console.log("a=",a)
console.log("++a=",++a) //pre increment
console.log("a=",a)
console.log("--a=",--a) //pre decrement
console.log("a=",a)
```

Post increment or decrement operators first used value of variable then increment or decrement them
Ex: -

```
let a=45;
console.log("a=",a)
console.log("a++=",a++) //post increment
console.log("a=",a)
console.log("a--=",a--) //post decrement
console.log("a=",a)
```

Comparison operators: - Comparison operators are used in logical statements to determine equality or relations between variables or values.

Operator	Description	Example
==	Is equal to	10=="10" = true
===	Identical (equal and of same type)	10==="10" = 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 operators: - Logical operators are used to determine the logic between variables or values.

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

Ex: -

```
let a=45, b=10;
let c=(a>b)&&(a==45)
console.log(`a=${a}, b=${b}, (a>b)&&(a==45)=${c}`)
```

Bitwise operators: - these operators convert number into binary (32bit integer) and perform logical operations on these numbers and result is converted into decimal numbers

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0000 0101	1111 1010	-128+64+32+16+8+0+2+0 =-6
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2

Ex: -

```
let a=5, b=6;
let c=a&b;
console.log(`a=${a}, b=${b}, a&b=${c}`)
```

Bitwise negation: - it first converts number in binary then invert bits of number, if first bit is 1 then take that bit number as negative and it is 0 then positive and convert it to decimal with 2's power

Ex: -

5= 0000 0101

~5= 1111 1010

It first bit is 1 then its number will be negative

- 1 1 1 1 1 0 1 0
- $(1*2^7) + (1*2^6) + (1*2^5) + (1*2^4) + (1*2^3) + (0*2^2) + (1*2^1) + (0*2^0)$
- 1*128 + 1*64 + 1*32 + 1*16 + 1*8 + 0*4 + 1*2 + 0*1
- 128 + 64 + 32 + 16 + 8 + 0 + 2 + 0
- 128 + 122
- 6

~5= -6

Assignment operators: - Assignment operators assign values to JavaScript variables.

Operator	Description	Example
=	Assign	Var a=10; Now a = 10
+=	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
**=	Exponentiation and assign	var a=10; a**=2; Now a = 100

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

String operators: - these operators compare or concatenate string

String comparison: - these operators compare string alphabetically

Operator	Description	Example
S1<S2	S1 before S2	"A"<"a" = true
S1>S2	S1 after S2	"a">"b" = false
S1==S2	S1 same as S2	"A"=="a" false

Ex: -
let s1="a", s2="b";
let r=s1<s2;
console.log(`s1=${s1}, s2=${s2}, s1<s2=${r}`)

String concatenate: - use "+" operator for string concatenation

Ex: - add to strings
let s1="hello"
let s2="world"
let s3=s1+" "+s2
console.log(s3)

Ex: - join number with string
let s1="hello"
let n2=5
let s3=s1+n2
console.log(s3)

Type operators: - it is used for identify types of variables and objects

Typeof: - it display type of object or variable
Syntax: - typeof data

Ex: -
```
let n1=45
let type=typeof n1
console.log(`n1=${n1}, type=${type}`)
```

Instanceof: - it return true if object is instance of an object type, else false
Syntax: - object instanceof object-type

Ex: -
```
let instance=n1 instanceof Number
console.log(`n1=${n1}, type=${type}`)
console.log(`n1 instanceof Number=${instance}`)
```

TYPE CONVERSION: - convert values to another data-type

TYPES OF TYPE CONVERSION: - there are two types for type conversion is implicit (automatic) and explicit (manually) conversion.

Implicit conversion: - This occurs when JavaScript automatically converts one data type to another.
Ex: -
```
let result;
result = "3" + 2; //  returns "32"
result = "3" + true; // returns "3true"
result = "3" + null; //  returns "3null"
result = 5 + null; // returns 5  ~null is converted to 0
result = true + true; // returns 2  ~true converted to 1
result = "3" - 2; //  returns 1  ~"3" converted to 3
```

Note: - In automatic conversion, JavaScript tries to convert values to the appropriate type when performing operations, which can lead to unexpected results if not understood.

Explicit conversion: - This is when you manually convert a value from one type to another using built-in function.
Ex: -
```
let result;
result = Number("5"); // 5
result = String(true); // "true"
result = Boolean(0); // false
```

TO NUMBER CONVERSION: - convert values to number

String to number: - convert strings to numbers

Number: - this is global method for convert values into number
Syntax: - Number("string")
Ex: -
let v1=Number("3.45") //return 3.45
let v2=Number(" ") // return 0
let v2=Number("good") //return NaN

NaN: - it is value for define unreprentable or undefined value in number, it known as “Not a Number”
Ex: -
let n=NaN
console.log(n)

parseFloat: - parse string and return floating point number
Ex: -
let n=parseFloat("45.56 is weight")
console.log(`float:- ${n}`)

parseInt: - parse string and return integer number
Ex: -
let n=parseInt("84.45Rs")
console.log(`Int:- ${n}`)

Unary + operator: - it convert variable to number
Ex: -
let d="5.5"
let n1=+d
console.log(n1)

Date to number: - convert date to number
Ex: - it returns timestamp of date
let d=new Date(); //define date object
console.log(d)
console.log(Number(d))

Boolean to number: - convert Boolean values to number
Ex: -
console.log(Number(true)) //return 1
console.log(Number(false)) //return 0

TO STRING CONVERSION: - convert values to string

Number to string: - convert number to string

String: - this global method converts number to string
Syntax: - String(Number)
Ex: -
let x=45
let str1=String(x)
console.log(str1)

Ex: -
console.log(String(34+45))

toString: - this method also convert value to string
Syntax: - value.toString()
Ex: -
let x=45
let str1=x.toString()
console.log(str1)

Boolean to string: - convert Boolean to string, using toString and String methods

String: -
Ex: -
let bool=true
let str1=String(bool)
console.log(str1)

toString: -
Ex: -
let bool=false
let str1=bool.toString()
console.log(str1)

Date to string: - convert date to string, using toString and String methods

String: -
Ex: -
let d=Date()
let str1=String(d)
console.log(str1)

toString: -
Ex: -
let d=Date()
let str1=d.toString()
console.log(str1)

TO BOOLEAN CONVERSION: - convert values into Boolean

Number to Boolean: - convert number to Boolean, use Boolean function for it.

Syntax: - Boolean(value)
Ex: -
```
let x=45
let bool=Boolean(x)
console.log(bool)
```

Ex2: -
```
let x=0
let bool=Boolean(x)
console.log(bool)
```

String to Boolean: - convert string to Boolean, use Boolean function for it.

Syntax: - Boolean(value)
Ex: -
```
let str1="string"
let bool=Boolean(str1)
console.log(bool)
```

Ex2: -
```
let str2=""
let bool=Boolean(str2)
console.log(bool)
```

BOOLEAN: - it is JavaScript data-type, which can take only two values true or false

BOOLEAN FUNCTION: - it is used for convert values to Boolean type or finds if expression is true or false

Syntax: -
Boolean(value)

Ex: -
```
let bool=Boolean(42>4)
console.log(`42>4= ${bool}`)
```

Comparisons and conditions: - The Boolean value of an expression is the basis for all JavaScript comparisons and conditions.

Ex: -
```
bool=Boolean(42>4)
bool=Boolean(42>4 && 34<100)
```

BOOLEAN VALUES: - there are two values in Boolean as true and false, other types of values are convert into these two values with Boolean conversion

True values: - all types with values converted into true

Values: -

Boolean(100)
Boolean(3.45)
Boolean(-35)
Boolean("hello")
Boolean("false")
Boolean(" ")
Boolean(Infinity)
Boolean(-Infinity)
Boolean(true)

False values: - all types without values converted into false

Values: -

Boolean()
Boolean(0)
Boolean(-0)
Boolean("")
Boolean(undefined)
Boolean(null)
Boolean(false)
Boolean(NaN)

BOOLEAN OBJECT: - Booleans are also defined as object with new keyword

Syntax: -

Var1=new Boolean(true)

Ex: -

```
let g=new Boolean(true)
console.log(g)
```

Note: - The new keyword complicates the code and slows down execution speed. Boolean objects can produce unexpected results

Comparing: - comparing Boolean object and literal, the type of Boolean object is object and literal's is Boolean

Ex: -

```
let b1=true;
let b2=new Boolean(true);
console.log(`typeof b1= ${typeof b1}`);
console.log(`typeof b2= ${typeof b2}`);
```

Comparing Boolean literal and object with equal and identical operators: -
Ex: -

```
let b1=true;
let b2=new Boolean(true);
let r=b1==b2; // return true
console.log(`b1==b2= ${r}`);
r=b1===b2; // return false
console.log(`b1===b2= ${r}`);
```

Comparing Boolean objects with equal and identical operators: -
Ex: -

```
let b1=new Boolean(true);
let b2=new Boolean(true);
let r=b1==b2; // return false
console.log(`b1==b2= ${r}`);
r=b1===b2; // return false
console.log(`b1===b2= ${r}`);
```

COMPARISONS: - it compares values with each other

COMPARISON OPERATORS: - this operators used for compare values with each others, it also called relational operators

Operator	Description	Example
==	Is equal to	10=="10" = true
===	Identical (equal and of same type)	10==="10" = 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

Types of equality: -

Loose equality: - Compares two values for equality after converting them to a common type.
Operator: - ==
Ex: -

```
let a = 5;
let b = '5';
console.log(a == b);  // true
```

Strict equality: - Compares two values for equality without type conversion. Both value and type must match
Operator: - ===
Ex: -

```
let a = 5;
let b = '5';
console.log(a === b);  // false
console.log(a === 5);  // true
```

Types of inequality: -

Loose inequality: - Compares two values for inequality after type conversion
Operator: - !=
Ex: -

```
let a = 5;
let b = '6';
console.log(a != b);  // true
```

Strict inequality: - Compares two values for inequality without type conversion.
Operator: - !==
Ex: -

```
let a = 5;
let b = '5';
console.log(a !== b);  // true
console.log(a !== 5);  // false
```

COMPARE DIFFERENT VALUE TYPES: - compare different types of values with each other

Value1	Operator	Value2	Result	Explanation
"12"	>	2	true	It convert string value to number, returns true
"12"	>	"2"	false	With both values string it compare alphabetically and by position (index), returns false
"ram"	<	"12"	False	
"ram"	>	12	False	It convert string in number as alphabet to NaN that NaN compare with number, return false
"ram"	<	12	False	
"ram"	>	"12"	True	With both values string it compare alphabetically and by position (index), returns true
"a"	>	"A"	True	
NaN	>	12	False	NaN compare with number, return false
NaN	<	12	False	

CONDITIONS: - it check conditions with multiple values and comparisons

LOGICAL OPERATORS: - these are used for check conditions with multiple comparisons

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

Example: -

```
let age=23;
let voterId=true;
let eligible=(age>18) && (voterId==true);
console.log(`Age=${age}, Voter-Id=${voterId}`);
console.log(`eligible=${eligible}`);
```

CONDITIONAL OPERATOR: - this operator assigns a value to a variable based on some condition, it also called ternary operator

Syntax: - condition? value1: value2;
Explain: - it returns value between question mark and colon If condition is true, otherwise returns value after the colon sign.

Ex: -

```
let a=45,b=42;
let c=a>b?"a is greater":"b is greater";
console.log(`a=${a}, b=${b}, result= ${c}`)
```

Nullish coalescing operator: - this operator return second argument if first is nullish (null or undefined)

Syntax: - var1=arg1??arg2;
Ex: -

```
let a=null;
let b="Good";
let c=a??b;
console.log("c=",c);
```

IF STATEMENT: - this statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax: -

```
if (condition)
{
    //block of statements if condition is true
}
```

Example:-

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

ELSE STATEMENT: - this statement to specify a block of code to be executed if the condition is false.

Syntax: -

```
if (condition)
{
    //block of statements if condition is true
}
else
{
    //block of statements if condition is false
}
```

Example: -

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

NESTED IF ELSE: - it use nested if else or nested if statements

Syntax: -

```
 if (condition) //outer if statement
{
    //block of statements if outer condition is true
    if(condition) //inner if statement
    {
        //block of statements if inner condition is true
    }
    else
    {
        //block of statements if inner condition is false
    }

} //outer else statement
else
{
    //block of statements if outer condition is false
}
```

Example: -

```
let a=90;
if(a>50){
    console.log("a is greater than 50");
    if(a>100){
        console.log("a is greater invalid");
```

```
    }
}
else{
    console.log("a is less than 50");
}
```

ELSE IF LADDER: - this statement to specify a new condition if the first condition is false.

Syntax: -

```
if (condition1)
{
    //block of statements if condition1 is true
}
else if (condition2)
{
    //block of statements if condition1 is false and condition2 is true
}
…
…
…
else if (conditionN)
{
    //block of statements if conditionN-1 is false and condition-N is true
}
else
{
    //block of statements if all above conditions are false
}
```

Example: -

```
let points=45.56;
if(points>50){
    console.log("class A");
}
else if(points>40){
    console.log("class B");
}
else if(points>30){
    console.log("class C");
}
else{
    console.log("class D");
}
```

SWITCH: - this statement selects one of many code blocks to be executed.

Syntax: -

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

Explain: -

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example:-

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

Break: - this keyword breaks out of the switch block. This will stop the execution inside the switch block.

Default: - this keyword specifies the code to run if there is no matched case.

Strict comparison: - Switch cases use strict comparison. The values must be of the same type to match.

COMMON CODE BLOCKS: - use same code for different switch cases.

Define multiple cases and write code at end of after all cases defined, and use break keyword after code.

Example: -

```
let day="monday";

switch(day){
    case "monday":
    case "tuesday":
    case "wednesday":
    case "thursday":
    case "friday":
        console.log("Work Days");
        break;
    case "saturday":
    case "sunday":
        console.log("Holidays");
        break;
    default:
        console.log("Invalid day");
}
```

Case values type: - in JavaScript switch case can use multiple values as characters, strings, integers and float values

Ex: -

```
case 34:        //integer value
case "hello":   //string value
case 'a':       //character value
case 4.5:       //float value
```

LOOPS: - Loops offer a quick and easy way to do something repeatedly. It can execute a block of code a number of times, as long as a specified condition is true.

WHILE LOOP: - this loop loops through a block of code as long as a specified condition is true, it check condition first then execute block of code that's why it also called entry controlled loop

Syntax: -

```
initialization
while (condition)
{
    //block of code
    Increment / decrement
}
```

Example: -

```
let a=1;
while(a<10){
    document.write("hello ",a,"<br>");
```

```
    a++;
}
```

DO WHILE LOOP: - this is exit control loop, this execute block of code first then check condition, then it will repeat the loop as long as the condition is true.

Syntax: -

```
initialization
do
{
    //block of code
    Increment / decrement
}
while(condition);
```

Example: -

```
let a=1;
do{
    document.write(a,”<br>”);
    a++;
}while(a<=10);
```

FOR LOOP: - it takes three expressions for execute block of code, it used for execute block of code for particular times.

Syntax: -

```
for (expression 1; expression 2; expression 3)
{
    //block of code
}
```

Expression 1: - is executed (one time) before the execution of the code block.

Expression 2: - defines the condition for executing the code block. If condition returns true then it keeps executing block of code and stop executing when condition return false.

Expression 3: - is executed (every time) after the code block has been executed.

Example: -

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

NESTED LOOP: - in JavaScript can also use nested loop, as for in for, while in while or while in do-while etc.

Syntax: -

```
loop1{ //outer loop
    block of code
    loop2{ //inner loop
        block of code
    }
}
```

Example: -

```
for(let i=1;i<=10;i++) //outer loop
{
    for(let j=1;j<=10;j++) //inner loop
    {
        console.log(`i=${i}, j=${j} `);
    }
}
```

Example2: - pattern printing

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

BREAK STATEMENT: - break statement used for jump out of loop and continue next program.

Keyword: - break

Example: -

```
for(i=0;i<10;i++)
{
    console.log(i);
    if(i==4)
    {
        break;
    }
}
```

CONTINUE STATEMENT: - this statement break (skip) current iteration of loop and start next iteration.

Keyword: - continue

Example: -

```
for(i=0;i<10;i++)
{
    if(i==4)
    {
        break;
    }
    console.log(i);
}
```

FUNCTIONS: - it is a block of code designed to perform a particular task

ADVANTAGE: -

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.

SYNTAX: -

Define function: - It use **function** keyword to define function, after keyword function name and parentheses "()" after function name and function body using curly braces "{}".

```
function function_name()
{
  //block of code
}
```

Parameters: - use parentheses after function name to define parameters in function and use comma for separate multiple defined parameters.

```
function function_name(parameter1, parameter2, parameter3,…, parameterN)
{
  //block of code
}
```

Return value: - use **return** keyword for return value from function.

```
function function_name()
{
  //block of code
  return value;
}
```

Function call: - call (invoke) function using function_name and parentheses after function also pass argument to function in parentheses when it called if neccesssary.

```
function_name();
function_name(val1,val2);
```

TYPES OF FUNCTION: - there are four types of functions

Function without argument and return value: - this type of function does not take any argument and also not return any value.

Syntax: -

```
function function_name()
{
    //block of code
}
```

Example: -

```
function greet() //function define
{
    console.log("good day");
}

greet(); //function calling
```

Function with argument and without return value: - this type of functions takes argument and not returns any value.

Syntax: -

```
function function_name(parameter1,parameter2)
{
    //block of code
}
```

Example: -

```
function greetname(name)
{
    console.log("good day "+name);
}
greetname("ram"); //function calling
```

Parameters: - parameters at function define are called **formal parameters** and arguments that passed to function invoking time is called **actual parameters**.

Ex: -

```
function greetname(name) //formal parameter
{
    console.log("good day "+name);
```

```
}
greetname("ram"); //actual parameter
```

Default argument: - JavaScript allow set default value to parameter
Ex: -

```
function greetname(name="") //default value
{
    console.log("good day "+name);
}
greetname();
```

Function without argument and with return value: - this type of functions does not take any argument but returns value. Use **return** keyword for return value.

Syntax: -

```
function function_name()
{
    //block of code
    return value;
}
```

Example: -

```
function getkey()
{
    let key="2347fgkg9834";
    return key;
}

let code=getkey();
console.log("code:-",code);
```

Assign return values to variables: - assign return value of function to variable using assignment operator.

Syntax: -

```
var1=function_name();
```

Ex: -

```
let code=getkey();
console.log("code:-",code);
```

Function with argument and with return value: - this type of functions takes any arguments and also returns value. Use **return** keyword for return value.

Syntax: -

```
function function_name(parameter1, …, parameter2)
{
```

```
    //block of code
    return value;
}
```

Example: -

```
function add(a,b)
{
    let ans=a+b;
    return ans;
}

let result=add(3,4);
console.log("additon:- ",result);
```

Pass variable's values to function: - in JavaScript also can pass variables values to function's arguments when calling it.

Syntax: -

```
var1, var2;
var3=function_name(var1,var2);
```

Ex: -

```
let x=34,y=67;
let z=add(x,y);
console.log("additon:- ",z);
```

Recursion: - when function calls itself is called recursion. Recursive function need to define base condition otherwise it will get in infinity.
Advantage: - it will reduce line of codes in function
Disadvantage: - it takes more time and memory to execute code.

Syntax: -

Example: -

```
function good(i,n=1){

    if(n<=i)
    {
        console.log(n);
        good(i,n+1);
    }
}
good(10);
```

Function advanced: -

Assigning anonymous function to variable: - define anonymous function and assign it to variable and use variable as function.

Syntax: -

```
Var1=function(parameters){
    //block of code
    return value;
}

Var1(paramters);
```

Example: -

```
let good=function(name){
    return "good day "+name;
}

let r=good("ram");
console.log(r);
```

Passing function as parameter: - pass function as parameter to another function, using function name and invoke it in another function. When function is passed as argument that function is called **callback function**

Syntax: -

```
Function1(){
    //block of code
}

Function2(funcparam){
    //block of code
    funcparam();
}

Function2(Function1);
```

Example: -

```
function good(){
    console.log("good day\n");
}

function operate(func1){
    func1();  //invoke passed function
}

operate(good);
```

Return function as value: - return function from another function.

Syntax: -

```
func1(){
    return func2(){
        //block of code
    }
}
Var1=func1();
Var1();
```

Example: -

```
function operate(){

return function(){
    console.log("good day");
    }
}

let good=operate(); //assign function to another value.
good();

operate()(); //calling returning function
```

Arrow function: - define function with arrow operator in JavaScript.

Syntax: -

```
Var1=(parameters)=>{
    //block of code
    return value;
}

Var2=Var1(argumnets);
```

Example: -

```
let good=(name)=>{
    console.log(`good day ${name}`);
}

good("ram");
```

Arrow function return value by default: - arrow function return value by default, if it not wrapped with curly braces and without semicolon. It does not required parentheses if there is single parameter

Syntax: -

```
var1=(parameters)=>n*n;
Var2=Var1(argumnets);
```

Example: -
```
let square=n=>n*n;
console.log(square(4));
```

Immediate invoking function: - invoke functions immediately after defining it, wrap function in parentheses and define parentheses after function wrapping parentheses

Syntax: -
```
(func1(){
    //block of code
})();
```

Example: -
```
(function(){
    console.log("good day to all");
})();
```

ARRAYS: - Array is an object that represents a collection of similar type of elements. It is homogeneous data structure. It takes index from 0, but in JavaScript array can store values of multiple data types in same array. In JavaScript array behave like heterogeneous data structure.

Declaration: -

Declare array literal: - declare array as literal with square brackets and values in it, separated by comma.

Syntax: -
```
Var1= [value1, …, valueN];
```

Example: -
```
const arr1=[12,34,5,55,67];
console.log(arr1);
```

Declare with new keyword: - declare array with new keyword and array method.

Syntax: -
```
Var1 = new Array(value1, …, valueN);
```

Example: -
```
const arr1=new Array(12,34,5,55,67);
console.log(arr1);
```

Declare empty array with size: - declare array with undefined (empty) element of given size.
Syntax: -
```
Var1=new Array(size);
```

Example: -
```
const arr1=new Array(10);
console.log(arr1);
```

Assigning: -

Assigning value to empty array literal: - declare empty array literal and then assign value by using index.

Syntax: -
```
Var1= [];
Var1[0]=value1;
Var1[1]=value2;
Var1[2]=value3;
```

Example: -
```
const arr1=[];
arr1[0]=45;
arr1[1]=25;
arr1[2]=85;

console.log(arr1);
```

Assigning value to empty new array: - declare empty array with new keyword and assign value using index.

Syntax: -
```
Var1= new Array();
Var1[0]=value1;
Var1[1]=value2;
Var1[2]=value3;
```

Example: -
```
const arr1=new Array();
arr1[0]=45;
arr1[1]=25;
arr1[2]=85;

console.log(arr1);
```

Accessing: - access array values using index for assigning new value to particular index or use it to other operations

Syntax: -
```
Var1[index]=value;
Var2=var1[index];
```

Example: -

```
const arr1=[34,55,33,22,39];
console.log(arr1);

arr1[3]=111; //assign new value to particular index of array
console.log(arr1);
console.log("arr[3]:-",arr1[3]); //print particular array element using index
```

Traverse array: - use loop for traversing array

For … of loop: - this loop traverse through every element of array sequentially.

Syntax: -

```
for(var1 of array1)
{
    //block of code
}
```

Example: -

```
const arr1=[34,55,33,22,39];
for(let i of arr1)
{
    console.log(i);
}
```

Nested Array: - define array in array as multidimensional arrays. It is also called array of an array or 2d array.

Syntax: -

```
Var1 = [[array1], [array2], [array3]];
```

Example: -

```
let ar2d=[[12,34,45],[24,56,67],[34,45,67]];
console.log(ar2d);
```

Accessing nested array: - access nested array using double index of array

Syntax: -

```
Var1[index][index]=val1
```

Example: -

```
console.log(ar2d[1][1]);
```

Traversing nested array: - use nested loop for traverse through nested array

Syntax: -

```
for(array1d of array2d)
{
    for(array1d of array2d)
    {
        //block of code
    }
}
```

Example: -

```
let ar2d=[[12,34,45],[24,56,67],[34,45,67]];
for(let arr of ar2d)
{
    for(let i of arr)
    {
        console.log(i);
    }
    console.log("\n");
}
```

Array methods: - there are build-in method for manipulate array data.

Length: - it returns length of array. It is property of array
Syntax: - arrayName.length
Example: - console.log(arr1.length);

toString(): - it convert array element in string with comma separated values.
Syntax: - arrayName.toString();
Example: -

```
v=arr1.toString();
console.log(v);
```

at(): - it returns element at given position at passed to argument.
Syntax: - arrayName.at(index);
Example: -

```
v=arr1.at(7);
console.log(v);
```

join(): - this method also return array element as string with provided string separator, separate with comma if not separator provided
Syntax: - arrayName.join("separator");
Example: -

```
v=arr1.join("/");
console.log(v);
```

pop(): - this method remove last element from array and returns that value.

Syntax: - arrayName.pop();
Example: -
v=arr1.pop();
console.log(v);
console.log(arr1);

push(): - this method add element at last position and return new length of array.
Syntax: - arrayName.push(value);
Example: -
v=arr1.push(78);
console.log(v);
console.log(arr1);

shift(): - this method remove first array element and shift other elements to lower index And return removed value.
Syntax: - arrayName.shift();
Example: -
v=arr1.shift();
console.log(v);
console.log(arr1);

unshift(): - this method add provided value to beginning of array and unshift other elements and return new size of array.
Syntax: - arrayName.unshift(value);
Example: -
v=arr1.unshift(78);
console.log(v);
console.log(arr1);

concat(): - this method merge (concat) provided arrays' elements with array and return new array. It also takes string as argument. Multiple arguments also can provide to this function.
Syntax: - arrayName.concat(arrays/strings);
Example: -
const arr2=[12,11,22,45];
v=arr1.concat(arr2,["a","b","c"],"string_value");
console.log(v);
console.log(arr1);

copyWithin(): - this method copies array elements to another position within array, without changing length of array and this method overwrites the existing values.
Syntax: - arrayName.copyWithin(paste from(index), start from(index), to index-1);
arr1.copyWithin(paste from (index), start from(index)); //copy till end
Example: -
console.log("good",arr1);
arr1.copyWithin(3,1,4); // (up to index 3 (4-1))
console.log(arr1);

flat(): - this method creates new array with sub-array elements concatenated to a specified depth. Remove single dimension from multidimensional array.
Syntax: - arrayName.flat();
Example: -

```
arr2=[[12,34,465],[11,22,33],[124,56],[45,67]];
v=arr2.flat();
console.log(v);
```

flatMap(): - this method first maps all elements of an array then creates new array by flattening the array. This function takes function as argument with three optional parameters as value and index and source array
Syntax: -

```
arrayName.flatMap(function1)
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
arr2=v.flatMap((x)=>[x, x*10]);
console.log(arr2);
```

splice(): - this method used for add new items and remove elements from array and return them as array.
Syntax: - var1=arrayName.splice(S, R, P1, P2 …);
S=start index for input items
R=remove number of elements from index
P1, P2 etc. = rest parameters are for elements input in array
Example: -

```
console.log(arr1);
arr1.splice(7,0,444,555,666,777); //add new items in array at given index
v=arr1.splice(9,4); //remove and return items from given index
console.log(arr1);
console.log(v);
```

slice(): - this method return elements from array in range of passed indices or return all array from passed index if only one index is given. It return elements as new array and does not removed from source array
Syntax: - var1=arrayName.slice(start-index, to index-1);
Example: -

```
v=arr1.slice(2,5);  //return array with elements in given range (up to index 4(5-1))
console.log(v);

8v=arr1.slice(2); //return all array element from index to last index
console.log(v);
```

indexOf(): - this method searches an array for element value and return its index (position) in array. It also can passed optional value for starting index, it returns -1 if value does not found
Syntax: - var1=arrayName.indexOf(value, start index(optional));
Example: -

```
v=arr1.indexOf(444);
console.log(v);

v=arr1.indexOf(444,8);  //starting index for finding value in array
console.log(v);
```

lastIndexOf(): - this method searches an array from backward direction for element value and return its index (last occurrence position) in array. It also can passed optional value for starting index, it returns -1 if value does not found
Syntax: - var1=arrayName.lastIndexOf(value, start index(optional));
Example: -

```
v=arr1.lastIndexOf(444);
console.log(v);

v=arr1.lastIndexOf(444,9);  //starting index for finding value in array at backward direction
console.log(v);
```

Note: - indexOf and lastIndexOf methods cannot find NaN value

Includes(): - this method check if element is present or not in array (including NaN)
Syntax: - var1=arrayName.includes(value);
Example: -

```
v=arr1.includes(NaN);
console.log(v);
```

find(): - this method returns the value of first array element that passes a test function, function should return value
Syntax: -

```
arrayName.find(function1)
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
v=arr1.find(function(val,ind,ar){
    if(val%2==0)
    {
        return val;
    }
});

console.log(v);
```

findIndex(): - this method returns the index of first array element that passes a test function, function should return value

Syntax: -

```
arrayName.findIndex(function1)
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
v=arr1.findIndex(function(val,ind,ar){
    if(val%2==0)
    {
        return val;
    }
});

console.log(v);
```

findLast(): - this method returns the value of last array element that passes a test function, function should return value

Syntax: -

```
arrayName.findLast(function1)
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
v=arr1.findLast(function(val,ind,ar){
    if(val%2==0)
    {
        return val;
    }
});

console.log(v);
```

findIndexLast(): - this method returns the index of last array element that passes a test function, function should return value

Syntax: -

```
arrayName.findIndexLast(function1)
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
v=arr1.findIndexLast(function(val,ind,ar){
```

```
    if(val%2==0)
    {
        return val;
    }
});

console.log(v);
```

sort(): - this method sort elements alphabetically in ascending order
Syntax: - arrayName.sort();
Example: -

```
arr1.sort();
console.log(arr1);
```

sort(function): - sort optionally take function as argument function should define with two parameters, for value comparison.
Function: - function(a, b) {return value;}
Return value: -

- If return value of this function is greater than 0 then it will sort 'a' argument data after 'b'
- If return value of this function is less than 0 then it will sort 'a' argument data before 'b'
- If return value of this function is equal to 0 or NaN then it will keep original order of arguments

Syntax: -

```
arrayName.sort(function1);
function function1(a,b){
    return value;
}
```

Example: - sort numerical values.

```
console.log(arr1);
arr1.sort((a, b)=>{
    return a-b;
});
console.log(arr1);
```

reverse(): - this method reverse elements of array
Syntax: - arrayName.reverse();
Example: -

```
arr1.reverse();
console.log(arr1);
```

forEach(): - this method calls a function for each elements in array, this function take three arguments as value, index and array

Syntax: -

```
arrayName.forEach(function1)
```

```
function1(value,index,source_array){
    //block of code
}
```

Example: -

```
arr1.forEach(good);

function good(val,ind,arr){
    console.log(ind,val);
}
```

map(): - this method creates new array by performing a function on each array element, this method does not change original array, passed function to this method take three arguments as value, index and array

Syntax: -

```
Var1=arrayName.map(function1)
function1(value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.map(function(val,ind,arr){
    return val*2;
});
console.log(v);
```

flatMap(): - this method creates new array by performing a function on each array element, and flattening the result by one level, this method does not change original array, passed function to this method take three arguments as value, index and array

Syntax: -

```
Var1=arrayName.flatMap(function1)
function1(value,index,source_array){
    //block of code
    return value array;
}
```

Example: -

```
v=arr1.flatMap(function(val,ind,arr){
    return arr;
});
console.log(v);
```

filter(): - this method creates a new array with array elements that passed test, this method does not change original array, passed function to this method take three arguments as value, index and array

Syntax: -

```
Var1=arrayName.filter(function1)
function1(value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.filter(function(val,ind,arr){
    if(typeof val=="number")
    {
        return val;
    }
});
console.log(v);
```

reduce(): - this method runs a function on each array element to produce a single value. This method does not change original array, passed function to this method take four arguments as initial/previously returned value, value, index and array

Syntax: -

```
Var1=arrayName.reduce(function1)
function1(initial/previously returned value,value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.reduce(function(total,val,ind,arr){
    return total+val;
});
console.log(v);
```

reduceRight(): - this method runs a function on each array element from right to left for produce a single value. This method does not change original array, passed function to this method take four arguments as initial/previously returned value, value, index and array

Syntax: -

```
Var1=arrayName.reduceRight(function1)
function1(initial/previously returned value,value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.reduceRight(function(total,val,ind,arr){
    return total+val;
});
console.log(v);
```

every(): - this value checks if all array elements are pass a test, This method does not change original array, passed function to this method take four arguments as value, index and array

Syntax: -

```
Var1=arrayName.every(function1)
function1(value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.every(function(val,ind,arr){
    return val > 18;
});
console.log(v);
```

some(): - this value checks if some array elements are pass a test, This method does not change original array, passed function to this method take four arguments as value, index and array

Syntax: -

```
Var1=arrayName.some(function1)
function1(value,index,source_array){
    //block of code
    return value;
}
```

Example: -

```
v=arr1.some(function(val,ind,arr){
    return val > 18;
});
console.log(v);
```

keys(): - this method return array iterator object with keys of an array

Syntax: - var1=arrayName.keys();

Example: -

```
v=arr1.keys();
console.log(v);

for(let x of v){
    console.log(x);
```

```
}
```

entries(): - this method return array iterator object with key/value pairs
Syntax: - var1=arrayName.entries();
Example: -

```
v=arr1.entries();
console.log(v);

for(let x of v){
    console.log(x);
}
```

fill(): - this method fill elements in array with static value.
Syntax: - arrayName.fill(value);
Example: -

```
arr1.fill(34);
console.log(arr1);
```

Array.from(): - this method returns an array object from any object with length property or iterable object.
Syntax: - var1=Array.from(value);
Example: -

```
name="ram";
v=Array.from(name);
console.log(v);
```

Array.isArray(): - this method check whether passed object is array or not.
Syntax: - var1=Array.isArray(value);
Example: -

```
name="ram";
v=Array.isArray(name);
console.log(v);
```

Array.of(): - this method convert multiple passed values to array.
Syntax: - var1=Array.of(value1, value2, …, valueN);
Example: -

```
v=Array.of(34,456,67,89,89,56);
console.log(v);
```

Array.prototype: - this property allows to adds new properties and methods to array

Syntax: - Array.prototype.name=value;
Example: -

```
Array.prototype.print=function(){
    console.log("print:-",this);
}
```

```
arr1=[34,45,67,78,93];
arr1.print();
```

this keyword: - this keyword is reference object that initialize or define function. Using this keyword, it can access value of that object; this keyword refers to different objects depending on how it is used, when used in an array method it refers to the array.

Assign method to particular array object: -

```
arr1=[34,45,67,78,93];
arr1.printf=function(){
    console.log("printf:-",this);
}
arr1.printf();

arr2=[23,34,56,67];
arr2.printf(); //this will get error for not a function
```

this keyword in arrow function: - this keyword in arrow function reference to window object.
Example: -

```
arr1=[34,45,67,78,93];
arr1.printf=()=>{
    console.log("printf:-",this);
}
arr1.printf();
```

Spread operator: - this operator expands iterable into more elements

Syntax: - var1=[…arr1,…arr2,…arrN]
Example: -

```
a1=["ram","shyam"];
a2=["madhav","keshav"];

a3=[...a1,...a2];
console.log(a3);
```

Passing multiple arguments to function using spread operator (rest parameter): - it is also called rest parameter of function.

Syntax: -

```
function functionName(…param1){ //defining function
    //block of code
}

functionName(arg1, arg2, …, argN); //calling function
```

Example: -

```
function good(...a){
    console.log(a);
}

good(34,45,67,78,89,89,65,45);
```

OBJECTS: - JavaScript has object data type that store data in key/value pairs and access it using keys. It is heterogeneous data structure. That can store multiple types of values.

Declaration: -

Declare object literal: - declare object literal using curly braces and keys and values in it, keys and values separated by colon (:) operator, and defined multiple key/value pairs separated by comma. These object literals also called object initializers.
Syntax: - var1={key1:value1, key2:value2, …, keyN:valueN};
Example: -

```
const person={name:"ram",age:23,gender:"Male"};
console.log(person);
```

example2: - it can also declare in multiple lines

```
const person={
    name:"ram",
    age:23,
    gender:"Male"
};

console.log(person);
```

Declare using new keyword: - object can also declare using new keyword, and assign key and values later.
Syntax: -

```
Var1=new Object();
Key1:value1;
Key2:value2;
…
KeyN:valueN;
```

Example: -

```
const car= new Object();
car.name="fiat";
car.model="M132";
car.color="red";

console.log(car);
```

Properties: - these are keys in JavaScript object for hold particular values; these are stored in object as unordered collections

Accessing properties: - object properties can access using dot notation and bracket notation (square brakets)

Syntax: -
Var1.propertyName="value";
Var1["propertyName"]="value";

Example: -
car.name="fiat";
console.log("car color is",car["color"]);

Adding new properties: - add new properties in object by giving values to it.

Syntax: -
Var1.newPropertyName="value";
Var1["newPropertyName"]="value";

Example: -
car.condition="good";
car["condition"]="good";

Delete properties: - object's properties can remove using delete keyword

Syntax: -
Var1.propertyName="value";
Var1["propertyName"]="value";

Example: -
delete car.name;
delete car["name"];
console.log(car);

Traverse object: -

For … in loop: - use this loop for traverse through loop by accessing its properties.
Syntax: -
```
for(let var1 in obj1)
{
    console.log(var1,":",obj1[var1]);
}
```

Example: -
for(let x in car)

```
{
    console.log(x,":",car[x]);
}
```

Nested object: - define nested object using object property to another object.

Define nested object: -

Syntax: -

```
Obj1={
    property1: value1,
    propertyObj: {
        propertyIn1: valueIn1,
        propertyIn2: valueIn2,
        propertyInN: valueInN,
    },
    property3: value3,
}
```

Example: -

```
const student={
    name:"ram",
    age:23,
    rollno:345,
    marks:{
        maths:45,
        pysics:75,
        chemistry:69,
        english:56
    }
}
```

Accessing nested object properties: - these properties can also access using dot notation and bracket notation

Syntax: -

```
Obj1.propertyObj.property=value
Obj1.propertyObj["property"]=value
Obj1["propertyObj"]["property"]=value
```

Example: -

```
console.log(student.marks.maths);
console.log(student.marks["physics"]);
console.log(student["marks"]["english"]);
```

Traverse nested object: - use nested for … in loop for traverse through nested object, also check that instance of property is object or not

Syntax: -

```
for(let var1 in obj1)
{
    if(obj1[var1] instanceof Object){
        for(let var2 in obj1[var1]){
            console.log(var1,var2,":",obj1[var1][var2]);
        }
    }
    else{
        console.log(var1,":",obj1[var1]);
    }
}
```

Example: -

```
for(let x in student){
    if(student[x] instanceof Object){
        for(let o in student[x]){
            console.log(x,o,":",student[x][o]);
        }
    }
    else{
        console.log(x,":",student[x]);
    }
}
```

Optional chaining operator: - this operator is defined with question mark and dot, it return undefined instead of error, if object is undefined or null.

Syntax: - objectOuter.objectInner?.property;

Example: -

```
const person={name:"ram",age:23,gender:"Male"};
console.log("Person car name:- ",person.car?.carname);
console.log("Person car name:- ",person.car.carname); //throw error for accessing undefined object's property
```

Methods: - these are actions that can be performed on objects, it is function definition stored as property value.

Syntax: -

```
Var1={
    property1: value1,
    property2:function(){
        //block of code
    }
}
```

Example: -

```
const data={
    firstname: "mark",
    lastname: "doe",
    fullname: function(){
        return this.firstname+" "+this.lastname;
    }
}

console.log(data.fullname());
```

this keyword: - The this keyword refers to different objects depending on how it is used, When used in an object method, this refers to the object.

Object static method: -

values(): - it returns an array of properties values of an object

Syntax: -

```
Var1=Object.values(obj1);
```

Example: -

```
v=Object.values(person);
console.log("values",v);
```

keys(): - it returns an array of properties of an object

Syntax: -

```
Var1=Object.keys(obj1);
```

Example: -

```
v=Object.keys(person);
console.log("keys",v);
```

entries(): - it returns an array of key/value pairs of an object.

Syntax: -

```
Var1=Object.entries(obj1);
```

Example: -

```
v=Object.entries(person);
console.log("entries",v);
```

defineProperty(): - it add property or change property value of object.

Syntax: -

```
Object.defineProperty(obj1,"propertyName",{value:"value"});
```

Example: -

```
Object.defineProperty(person,"contact",{value:"12345"});
```

descriptors: -
value: - it set value of property. Default value is undefined
writable: - it set property is writable or not. Default value is false
enumerable: - it set property for access in for..in loop or by methods. Default value is false.
configurable: - it enable delete for property from object. Default value is false.

Example: -

```
Object.defineProperty(person,"contact",{writable:true,enumerable:true,configurable:true});
console.log(person);

person.contact=11111;
console.log("edit value",person);

console.log("keys",Object.keys(person));

delete person.contact;
console.log("delete",person);
```

defineProperties(): - it adds multiple properties or change properties values of object.
Syntax: -

```
Object.defineProperties(obj1,{
    newProperty1:{value:"value1"},
    newProperty2:{value:"value2"}
});
```

Example: -

```
Object.defineProperties(person,{
    language:{value:"marathi"},
    email:{value:"ram123@gmail.com"}
});
```

Defineproperties with other descriptors: -

```
Object.defineProperties(person,{
    "email":{value:"ram123@gmail.com",writable:true,enumerable:true,configurable:true},
    "contact":{value:12345, writable:true, enumerable:true, configurable:true},
});
```

getOwnPropertyNames(): - this method returns array of properties from object.
Syntax: -

```
Var1=Object.getOwnPropertyNames(obj1);
```

Example: -

```
v=Object.getOwnPropertyNames(person);
console.log("names",v);
```

getOwnPropertyDescriptor(): - this method return descriptor of property from object.
Syntax: -
Var1=Object.getOwnPropertyDescriptor(obj1,"propertyName");

Example: -
v=Object.getOwnPropertyDescriptor(person,"contact");
console.log("contact descriptor",v);

getOwnPropertyDescriptors(): - this method return descriptor of all properties from object.
Syntax: -
Var1=Object.getOwnPropertyDescriptors(obj1);

Example: -
v=Object.getOwnPropertyDescriptors(person);
console.log(v);

fromEntries(): - this method return created object from iterable list (2d array) of key/values pairs
Syntax: -
Var1=Object.fromEntries([["key1","value1"],["key2",value2], … ,["keyN",valueN]]);

Example: -
v=Object.fromEntries([["grade","A+"],["percentage",90.45],["total",452]]);
console.log("from entries",v);

freeze(): - it prevent any change in object, it prevent add or delete properties or change values of properties.
Syntax: - Object.freeze(obj1);
Example: - Object.freeze(person);

isFrozen(): - it checks object is freeze or not, return true if object is freeze.
Syntax: -
Var1=Object.isFrozen(obj1);

Example: -
v=Object.isFrozen(person);
console.log("person is frozen",v);

preventExtensions(): - it prevent adding new properties in object.
Syntax: - Object.preventExtensions(obj1);
Example: - Object.preventExtensions(person);

isExtensible(): - it checks object is extensible or not, return true if object is extensible.
Syntax: -
Var1=Object.isExtensible(obj1);

Example: -

```
v=Object.isExtensible(person);
console.log("person is extensible",v);
```

seal(): - it prevent adding new or deleting existing properties in object.
Syntax: - Object.seal(obj1);
Example: - Object.seal(person);

isSealed(): - it checks object is sealed or not, return true if object is sealed.
Syntax: -

```
Var1=Object.isSealed(obj1);
```

Example: -

```
v=Object.isSealed(person);
console.log("person is sealed",v);
```

valueOf(): - this method return value of object
Syntax: - var1=obj1.valueOf();
Example: - v=person.valueOf();

create(): - this method create object from existing object.
Syntax: - obj2=Object.create(obj1);
Example: -

```
let person={name:"ram",age:23,gender:'M',intro:function(){
    console.log(`My name is ${this.name}, i am ${this.age} years old.`);
}};
person.intro();

let man=Object.create(person);
man.name="sham";
console.log(man);
man.intro();
```

assign(): - this method copies properties from one or more source objects to a target object.
Syntax: - Object.assign(obj1(target),obj2(sources));
Example: -

```
const person1 = {
    firstName: "ram",
    lastName: "roy",
    age: 50,
    eyeColor: "blue"
};
console.log(person1);

const person2 = {firstName: "sham",lastName: "goyal"};
console.log(person2);
```

```
Object.assign(person1, person2);
console.log(person1);
```

Assign multiple objects: - Object.assign(person1, person2, person3);

SCOPE: - Scope determines the accessibility (visibility) of variables.

Block scope: - variables defined with block scope keywords cannot access outside { } (curly braces) block.

Keywords: - 'const' and 'let' are these keywords for declare variables in block scope.

Example: -

```
let b=56;
{
    let b=34;
    console.log(`(inside block)b=${b}`); //34
}
console.log(`(outside block)b=${b}`); //56
```

Global scope: - variables declare outside block or functions are global scope variables.

Example: -

```
let b=56; //global scope
{
    console.log(`b=${b}`);
}
```

var keyword: - variables declared in block with var keyword are also global scope variables.

Example: -

```
{
    var v=45;
}
console.log(`v:- ${v}`);
```

Local/function scope: - variables defined inside function are function scope variables, that defined with var, let or const keywords.

Example: -

```
function good()
{
    var t=78;
}
good();
console.log(`t:- ${t}`); //this will cause error
```

Variables without scope keywords: - variables inside function define without any scope keywords are consider as global keywords

```
function good()
{
    t=78;
}
good();
console.log(`t:- ${t}`);
```

Function parameters: - variables declared in function parameters are local scope.

Example: -

```
function add(a,b)
{
    let c=a+b;
    console.log(`${a}+${b}=${c}`);
}

add(4,5);
console.log(`${a}, ${b}`);
```

let vs var: - variables with var keywords can be declared multiple times but with let keyword declaring same variable multiple time will throw syntax error

Example: -

```
var aa=45;
console.log(`aa:- ${aa}`);
var aa=5;
console.log(`aa:- ${aa}`);

let a=40;
console.log(`a:- ${a}`);
let a=5; //syntax error
console.log(`a:- ${a}`);
```

STRINGS: - these are for storing text, these are written with quotes.

String literal: - it defined inside quotes

Syntax: - var1="value";

Example: -

```
let str1="good day";
console.log(str1);
```

String quotes: - string can be defined in single or double quotes

Example: -

```
let str1="good day"; //double quotes String
let str2='hello world'; //single quotes String
```

Quotes inside quotes: - also can use quotes inside quotes
Example: -

```
var str1="good day 'ram'"; //single quotes inside double quotes
var str2='good day "ram"'; //double quotes inside single quotes
```

Escape characters: - use these characters for similar quoting in string or backslash.

Character	Uses
\'	Single quote
\"	Double quote
\\	Backslash

Example: -

```
var str1="good day \"ram\""; //double quotes escape character
console.log(str1);
var str2='hello world \'computer\\machine\''; //single quotes and backslash escape character
console.log(str2);
```

String object: - define string object using new keyword, but it slow down execution speed.

Syntax: - var1=new String("string_value");
Example: -

```
let strobj=new String("good day");
console.log(strobj);
```

Comparison of string literal and object: - it return true equal in value comparison but not in identical comparison.

Example: -

```
let x = "John";
let y = new String("John");
console.log("x:-",x);
console.log("y:-",y);
console.log("x==y:-",x==y); //equal comparison (true)
console.log("x===y:-",x===y); //identical comparison (false)
```

Type of string literal and object: -
Example: -

```
let x = "John";
let y = new String("John");
console.log("x:-",x);
console.log("y:-",y);
console.log("typeof x:- ",typeof x);
```

```
console.log("typeof y:- ",typeof y);
console.log("y instanceof String:- ",y instanceof String);
```

For…of loop in string: -

Example: -

```
let strobj=new String("good day");

for(let i of strobj)
{
    console.log(i);
}
```

String length: - this property return count of characters (length) in string.
Syntax: - var1=str1.length;

Example: -

```
let text=new String("good day");
console.log("text:-",text);
console.log("length of text:-",text.length);
```

Template strings: -

Declaration: - this strings enclosed with back-ticks, it allow single and double quotes inside the string.

Syntax: - var1=`string_value`;
Example: -

```
var str1=`He's often called "Ram"`;
console.log(str1);
```

Interpolation: - it is method of replacing variables and expressions to values in string. Template strings provide easy way to interpolation.
Syntax: - `${variable/expression} `;

Variable substitutions: -
Example: -

```
let name="ram";
let age=33;
let text=`My name is ${name}, i am ${age} years old.`;
console.log(text);
```

Expression substitutions: -
Example: -

```
let a=45;
let b=5;
let add=`Addition of ${a} and ${b} is ${a+b}.`;
```

```
console.log(add);
```

Html template: - use template to print html tags

```
let header = "Fruits list";
let tags = ["Apple", "Banana", "Cherry","Grapes","Mango"];

let html = `<h2>${header}</h2><ul>`;
for (const x of tags) {
  html += `<li>${x}</li>`;
}
html += `</ul>`;

document.write(html);
```

Multiline strings: - define multiline strings

Multiline template strings: - template strings can be defined in multiline.

Example: -

```
var str2=
`The quick
brown fox
jumps over
the lazy dog`;
console.log(str2);
```

Breaking long lines: - break long lines for program readability, break line using addition operator.

Example: -

```
let form1=
   "<form>"
+"<input type='text' placeholder='enter name'><br><br>"
+"<input type='submit' value='show'>"
+"</form>";

document.write(form1);
```

String methods: -

charAt: - this method return character at a specified index, it only take positive numbers.

Syntax: - var1=str1.charAt(index);

Example: -

```
let str1="Good_Day";
let ch=str1.charAt(0);
console.log("charAt(0):-",ch);
```

charCodeAt: - this method return character code UTF-16 (0 to 65,535) at a specified index.

Syntax: - var1=str1.charCodeAt(index);
Example: -
```
let str1="Good_Day";
let ch=str1.charCodeAt(0);
console.log("charCodeAt(0):-",ch);
```

at: - this method returns the character at specified index in string. It also takes negative numbers and return character from last index.
Syntax: - var1=str1.at(index); //positive/negative numbers
Example: -
```
let str1="Good_Day";
let ch=str1.at(-1);
console.log("at(-1):-",ch);
```

slice: - this method return extracted part in new string, it take two parameters start and end index (end index not included).
Syntax: - var1=str1.slice(start index, end index(not included));
Example1: -
```
let str1="Good_Day";
let ch=str1.slice(2,6);
console.log("slice(2,6):-",ch);
```

Example2: - only start index parameter, it will return all string from start parameter
```
let str1="Good_Day";
let ch=str1.slice(5);
console.log("slice(5):-",ch);
```

Example3: - assign negative value to only start index parameter; it will count position from end of string.
```
let str1="Good_Day";
let ch=str1.slice(-5);
console.log("slice(-5):-",ch);
```

Example4: - assign negative values to start and end parameter
```
let str1="Good_Day";
let ch=str1.slice(-3,-1);
console.log("slice(-3,-1):-",ch);
```

substring: - it extract string value from given start and stop index, if values is less than 0, then it will treated as zero.
Syntax: - var1=str1.substring(start index, end index(not included));
Example1: -
```
let str1="Good_Day";
let ch=str1.substring(-1,2);
console.log("substring(-1,2):-",ch);
```

example2: - if only one value passed, it return string from that value index to end of string

```
let str1="Good_Day";
let ch=str1.substring(5);
console.log("substring(5):-",ch);
```

substr: - this method extract part of string, with start index to specified length.
Syntax: - var1=str1.substr(start index, length for extract part);
Example1: -

```
let str1="Good_Day";
let ch=str1.substr(3,5);
console.log("substr(3,5):-",ch);
```

example2: - if only one value passed, it return string from that value index to end of string

```
let str1="Good_Day";
let ch=str1.substr(5);
console.log("substr(5):-",ch);
```

example3: - if negative index passed it will count from end of string, extract part in given length

```
let str1="Good_Day";
let ch=str1.substr(-5,3);
console.log("substr(-5,3):-",ch);
```

example4: - it will return string until it ends, if only negative index passed (count from end of string)

```
let str1="Good_Day";
let ch=str1.substr(-5);
console.log("substr(-5):-",ch);
```

toUpperCase: - it return string in uppercase letters.
Syntax: - var1=str1.toUpperCase();
Example: -

```
let str1="Good_Day";
let ch=str1.toUpperCase();
console.log("toUpperCase():-",ch);
```

toLowerCase: - it return string in lowercase letters.
Syntax: - var1=str1.toLowerCase();
Example: -

```
let str1="Good_Day";
let ch=str1.toLowerCase();
console.log("toLowerCase():-",ch);
```

concat: - it joins two or more strings.
Syntax: - var1=str1.concat(str2,str3,…);
Example: -

```
let str1="Good_Day";
let ch=str1.concat(" ","Ram");
console.log("concat(' ','Ram'):-",ch);
```

trim: - this method removes whitespaces from both side of string
Syntax: - var1=str1.trim();
Example: -
```
let str2="       \n     GOOD DAY    \t  \n ";
let ch=str2.trim();
console.log("trim():-",ch);
```

trimStart: - this method removes whitespaces from only start side of string
Syntax: - var1=str1.trimStart();
Example: -
```
let str2="       \n     GOOD DAY    \t  \n ";
let ch=str2.trimStart();
console.log("trimStart():-",ch);
```

trimEnd: - this method removes whitespaces from only end side of string
Syntax: - var1=str1.trimEnd();
Example: -
```
let str2="       \n     GOOD DAY    \t  \n ";
let ch=str2.trimEnd();
console.log("trimEnd():-",ch);
```

padStart: - this method pads string from the start. It pads string with another string multiple times until it reaches a given length. If pad length is less than string it will not show effects.
Syntax: - var1=str1.padStart(length,str2);
Example: -
```
let str1="Good_Day";
let ch=str1.padStart(10,"*");
console.log("padStart(10,'x'):-",ch);
```

padEnd: - this method pads string from the end. It pads string with another string multiple times until it reaches a given length. If pad length is less than string it will not show effects.
Syntax: - var1=str1.padStart(length,str2);
Example: -
```
let str1="Good_Day";
let ch=str1.padStart(10,"*");
console.log("padStart(10,'x'):-",ch);
```

repeat: - it return string with number of copies of string.
Syntax: - var1=str1.repeat(length);
Example: -
```
let str1="Good_Day";
let ch=str1.repeat(3);
console.log("repeat(3):-",ch);
```

replace: - it replace specified value with another string value. It will only replace string that will first find, and it is case sensitive.
Syntax: - var1=str1.replace("old string", "new string");
Example1: -

```
let str1="Good_Day";
ch=str1.replace("Day", "Night");
console.log("replace('Day', 'Night'):-",ch);
```

Example2: - use /i for replace case insensitive, it will replace only first found string, do not use quotes with /i statement
Syntax: - /newstring/i

```
let str1="Good_Day";
ch=str1.replace(/DAY/i, "Night");
console.log("replace('Day', 'Night'):-",ch);
```

Example3: - use /g for replace all matching string, do not use quotes with /g statement
Syntax: - /newstring/g

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.replace(/day/g,"Night");
console.log("replace(/day/g,'Night'):-",ch);
```

Example4: - use /gi for replace all matching case insensitive string, do not use quotes with /gi statement

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.replace(/day/gi,"Night");
console.log("replace(/day/gi,'Night'):-",ch);
```

replaceAll: - it replace all matching string with another string.
Example1: -

```
let str2="good day to all, here Day by day work hard, days will pass";
ch=str2.replaceAll("day","Night");
console.log("replaceAll('day','Night'):-",ch);
```

Example2: - use /gi for case insensitive matching, without g it will throw type error, do not use quotes with /gi statement

```
let str2="good day to all, here Day by day work hard, days will pass";
ch=str2.replaceAll(/day/gi,"Night");
console.log("replaceAll(/day/gi,'Night'):-",ch);
```

split: - it will split string into array from specified string or character
Syntax: - var1=str1.split("string");
Example: -

```
let str1="Good_Day";
let ch=str1.split("_");
console.log("split('_'):-",ch);
```

indexOf: - it return index of first occurrence of given string in string or return -1 if not found, it also accept string position for start searching.
Syntax: - var1=str1.indexOf("string");
Example: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.indexOf("day");
console.log("indexOf('day'):-",ch);
```

Syntax2: - var1=str1.indexOf("string",position);
Example2: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.indexOf("day",9);
console.log("indexOf('day',9):-",ch);
```

lastIndexOf: - it return index of last occurrence of given string in string or return -1 if not found. it also accept string position for start searching, and search backward;
Syntax: - var1=str1.lastIndexOf("string");
Example: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.lastIndexOf("day");
console.log("lastIndexOf('day'):-",ch);
```

Syntax2: - var1=str1.lastIndexOf("string",position);
Example2: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.lastIndexOf("day",40);
console.log("lastIndexOf('day',40):-",ch);
```

search: - this method return index of first occurrence of given string from string, and also can use /…/ syntax for search.
Syntax: - var1=str1.search("string");
Example1: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.search('day');
console.log("search('day'):-",ch);
```

Example2: - using /i for case insensitive searching

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.search(/DAY/i);
console.log("search(/DAY/i):-",ch);
```

match: - this method return array containing result for matching string from string. If match not found then it will return null.
Syntax: - var1=str1.match("string");
Example1: -

```
let str2="good day to all, here Day by day work hard, days will pass";
```

```
let ch=str2.match('day');
console.log("match('day'):-",ch);
```

Example2: - use /i statement for case insensitive match

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.match(/day/i);
console.log("match(/day/i):-",ch);
```

Example3: - use /g statement for global match

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.match(/day/g);
console.log("match(/day/g):-",ch);
```

Example4: - use /gi statement for global case-insensitive match

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.match(/day/gi);
console.log("match(/day/gi):-",ch);
```

matchAll: - this method return iterator containing the all results of matching values from string, use Array.from() method to convert iterator into array or spread operator[…iterator].

Syntax: - var1=str1.matchAll("string");

Example1: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let iter=str2.matchAll('day');
let ch=Array.from(iter);
//let ch=[…iter];
console.log("matchAll('day'):-",ch);
```

Example2: - use /g for match /…/ this statement, or it wll throw type error

```
let str2="good day to all, here Day by day work hard, days will pass";
let iter=str2.matchAll(/day/g);
let ch=Array.from(iter);
console.log("matchAll(/day/g):-",ch);
```

Example3: - use /gi for match case-insensitive values

```
let str2="good day to all, here Day by day work hard, days will pass";
let iter=str2.matchAll(/DAY/gi);
let ch=Array.from(iter);
console.log("matchAll(/DAY/gi):-",ch);
```

includes: - this method return true if string contains given value, it also take start position for search value.

Syntax1: - var1=str1.includes("string");

Example1: -

```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.includes("day");
console.log("includes('day'):-",ch);
```

Syntax2: - var1=str1.includes("string",index);
Example2: -
```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.includes("day",9);
console.log("includes('day',9):-",ch);
```

startsWith: - this method return true, if string start with given value or return false, it also take index for start value.
Syntax1: - var1=str1.startsWith("string");
Example1: -
```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.startsWith("good");
console.log("startsWith(good'):-",ch);
```

Syntax2: - var1=str1.startsWith("string",index);
Example2: -
```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.startsWith("day",5);
console.log("startsWith('day',5):-",ch);
```

endsWith: - this method return true, if string end with given value or return false, it also take number for check if first number of characters string ends with given value.
Syntax1: - var1=str1.endsWith("string");
Example1: -
```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.endsWith("good");
console.log("endsWith(good'):-",ch);
```

Syntax2: - var1=str1.endsWith("string",index);
Example2: -
```
let str2="good day to all, here Day by day work hard, days will pass";
let ch=str2.endsWith("day",8);
console.log("endsWith('day',8):-",ch);
```

codePointAt: - this method return Unicode, it can return full value of a Unicode value greater 0xffff (65535);
Syntax: - var1=str1.codePointAt(index);
Example: -
```
let str1="Good_Day";
let ch=str1.codePointAt(0);
console.log("codePointAt(0):-",ch);
```

localeCompare: - this method compare two strings in current locale (language) and return sort order as -1,0,1
Syntax: - var1=str1.localeCompare(compareString);

Order: -
-1: - if the string value is sorted before compareString value
0: - if the string and compareString have same value.
1: - if the string value is sorted after compareString value

Example: -
```
let s1= 'A';
let s2='B';
ch=s1.localeCompare(s2);
console.log("localeCompare(s2):-",ch);
```

fromCharCode: - this method return string from one or multiple Unicode.
Syntax: - var1=String.fromCharCode(n1,n2,n3,…,nX);
Example: -
```
let ch=String.fromCharCode(65,97,66,98,67,99);
console.log("String.fromCharCode(65,97,66,98,67,99):-",ch);
```

REGEXP: - A regular expression is a sequence of characters that forms a search pattern. It is used for search data in text by describing search pattern. It can be used to perform all types of text search and text replaces operations.

Syntax: - /pattern/modifiers
Example: - /GOOD/i
```
let pattern=/GOOD/i;
console.log("pattern:- ",pattern);
let str1="good day";
console.log("str1:- ",str1);
let res=str1.search(pattern);
console.log("str1.search(pattern):-",res);
```

RegExp: - this method converts string into regular expression if it is valid, and take optional argument for modifier
Syntax: - reg1= new RegExp("string","modifiers");
Example: -
```
let reg= new RegExp("good",'g');
console.log("RegExp:-",reg);
```

String methods: - use RegExp in search, replace methods of string.

Search method: -
Example: -
```
let pattern=/DAY/i;
console.log("pattern:- ",pattern);
let str1="good day";
console.log("str1:- ",str1);
```

```
let res=str1.search(pattern);
console.log("str1.search(pattern):-",res);
```

Replace method: -

```
let pattern=/DAY/i;
console.log("pattern:- ",pattern);
let str1="good day";
console.log("str1:- ",str1);
let res=str1.replace(pattern,"night");
console.log("str1.replace(pattern,'night'):-",res);
```

Note: - other methods as match, matchAll, search, replace, replaceAll, split can also use in regexp.

RegExp method: -

test: - this method return true if regexp pattern find match in string, else return false.
Syntax: - regexp.test("string");
Example: -

```
let reg=/day/
console.log("RegExp:-",reg);
let res=reg.test('good day');
console.log("reg.test('good day'):-",res);
```

exec: - it returns found text as object if regexp patern match in string, otherwise return null
Syntax: - regexp.exec("string");
Example: -

```
let reg=/good/
console.log("RegExp:-",reg);
let res=reg.exec('good day');
console.log("reg.exec('good day'):-",res);
```

Modifiers: - it specify matching area in string

g: - it specifies a global match, it find all matches in string
Syntax: - regex1=/pattern/g
Example: -

```
let reg=/y/g
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(str1):-",res);
```

i: - it specifies case-insensitive match.
Syntax: - regex1=/pattern/i

Example: -

```
let reg=/Y/gi
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(str1):-",res);
```

m: - it used for check pattern in multiline, it only affect with ^ and $ quantifiers for search pattern at start or end of every line in multiple lines string

Syntax: - regex1=/pattern/m

Example: -

```
let reg=/^good/gm
console.log("RegExp:-",reg);
let str1=`good day to all
good night to all
good bye`;
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

Quantifiers: - it define position or counts for pattern to match in string

^n: - it matches any string with n at beginning of it.

Syntax: - reg=/^n/

Example: -

```
let reg=/^good/
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let res=str1.match(reg);
console.log("str1.match(reg):-",res);
```

n$: - it matches any string that end with n.

Syntax: - reg=/n$/

Example: -

```
let reg=/all$/
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let res=str1.match(reg);
console.log("reg.match(str1):-",res);
```

n+: - it matches n in string at least one or multiple times

Syntax: - reg=/n+/
Example: -

```
let reg=/o+/g
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

n*: - it matches n in string zero or multiple times
Syntax: - reg=/n*/
Example: -

```
let reg=/da*/g
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

n?: - it matches any string that contains zero or one occurrences of n
Syntax: - reg=/…n?/
Example: -

```
let reg=/ay?/g
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

(?=n): - it matches any string that followed by a specific string n
Syntax: - reg=/…(?=n)/
Example: -

```
let reg=/good(?= day)/g
console.log("RegExp:-",reg);
let str1="good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

(?!n): - it matches any string that not followed by a specific string n
Syntax: - reg=/…(?!n)/
Example: -

```
let reg=/to(?!day)/g
console.log("RegExp:-",reg);
let str1="today, good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

n{X}: - it matches any string that contains a sequence of X n's. X must be a number
Syntax: - reg=/n{X}/
Example: -

```
let reg=/o{2}/g
console.log("RegExp:-",reg);
let str1="today, good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

n{X,Y}: - it matches any string that contains a sequence of X to Y n's. X and Y must be a number
Syntax: - reg=/n{X,Y}/
Example: -

```
let reg=/o{1,2}/g
console.log("RegExp:-",reg);
let str1="today, good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

n{X,}: - it matches any string that contains a sequence of at least X n's. X must be a number
Syntax: - reg=/n{X,}/
Example: -

```
let reg=/o{1,}/g
console.log("RegExp:-",reg);
let str1="today, good day to all";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

Groups: - use groups for create pattern for regexp

[abc]: - it match any character from bracket
Syntax: - regex1=/[characters]/
Example: -

```
let reg=/[abc]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

[A-Z]: - it matches any character in range.
Syntax: - regex1=/[char1-charN]/
Example: -

```
let reg=/[a-z]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

Other ranges: -
[a-z]: - any character in range from lowercase a to lowercase z
[A-Z]: - any character in range from uppercase Z to uppercase Z
[A-z]: - any character in range from uppercase A to lowercase z
[h-o]: -any character in range from lowercase h to lowercase o

[^abc]: - it match any character is not from bracket, use circumflex (^) operator for not include.
Syntax: - regex1=/[characters]/
Example: -

```
let reg=/[^abc]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

[^A-Z]: - it matches any character not in range, use circumflex (^) operator for not include
Syntax: - regex1=/[^char1-charN]/
Example: -

```
let reg=/[^a-z]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

Other ranges: -
[^a-z]: - not any character in range from lowercase a to lowercase z
[^A-Z]: - not any character in range from uppercase Z to uppercase Z
[^A-z]: - not any character in range from uppercase A to lowercase z

[^h-o]: -not any character in range from lowercase h to lowercase o

[123]: - it match any digit from bracket
Syntax: - regex1=/[digits]/
Example: -
```
let reg=/[123]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

[0-9]: - it matches any number in range.
Syntax: - regex1=/[digit1-digitN]/
Example: -
```
let reg=/[0-9]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

[^123]: - it matches any digit not from bracket
Syntax: - regex1=/[^digits]/
Example: -
```
let reg=/[^123]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

[^0-9]: - it matches any number not in range.
Syntax: - regex1=/[^digit1-digitN]/
Example: -
```
let reg=/[^0-9]/
console.log("RegExp:-",reg);
let str1="2 anybody";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

(red|green): - it match any from group alternative
Syntax: - regex1=/(alternate1|alternate2|…|alternateN)/
Example: -
```
let reg=/(red|green)/
```

```
console.log("RegExp:-",reg);
let str1="anybody see red or green color";
console.log("str1:-",str1);
let res=reg.exec(str1);
console.log("reg.exec(str1):-",res);
```

RegExp properties: - built in properties of regular expression

global: - it return true if RegExp have global modifier
Syntax: - var1=regex1.global;
Example: -

```
let reg=/good/g
console.log("RegExp:-",reg);
let res=reg.global;
console.log("reg.global:-",res);
```

ignoreCase: - it return true if RegExp have case-insensitive modifier
Syntax: - var1=regex1.ignoreCase;
Example: -

```
let reg=/good/i
console.log("RegExp:-",reg);
let res=reg.ignoreCase;
console.log("reg.ignoreCase:-",res);
```

multiline: - it return true if RegExp have multiline modifier
Syntax: - var1=regex1.multiline;
Example: -

```
let reg=/^good/gm
console.log("RegExp:-",reg);
let res=reg.multiline;
console.log("reg.multiline:-",res);
```

source: - it return text of RegExp pattern
Syntax: - var1=regex1.source;
Example: -

```
let reg=/o{2}/g
console.log("RegExp:-",reg);
let res=reg.source;
console.log("reg.source:-",res);
```

lastIndex: - it return index of match start of RegExp pattern after the last match found by test() or exec() methods. It only work with global modifier, if match not found then lastInndex reset to 0
Syntax: - var1=regex1.lastIndex;
Example: -

```
let reg=/o{1,}/g;
console.log("RegExp:-",reg);
```

```
let str1="today, good day to all";
console.log("str1:-",str1);
while(reg.test(str1)==true)
{
    let i=reg.lastIndex;
    console.log("match at index:- ",i);
}
```

Metacharacters: - these character used for special purpose, most of them defined with \

.(dot): - it matches any character except new line or other line terminator
Syntax: - regex1= /. /;
Example1: - global search for any 2 characters between g and d

```
let reg=/g..d/g
console.log("RegExp:-",reg);
let str1="your grade in examination is good";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

Example2: - global search for any 2 to 3 characters between g and d

```
let reg=/g.{2,3}d/g
console.log("RegExp:-",reg);
let str1="greed is never good";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\w: - it matches word characters as a-z, A-Z, 0-9 and _ (underscore), it defined as lowercase w
Syntax: - regex1= /\w/;
Example: -

```
let reg=/\w{8}/g
console.log("RegExp:-",reg);
let str1="good_123";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\W: - it matches non-word characters that not include a-z, A-Z, 0-9 and _ (underscore), it defined as uppercase W
Syntax: - regex1= /\W/;
Example: -

```
let reg=/\W/g
```

```
console.log("RegExp:-",reg);
let str1="Give it 100%";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\d: - it matches digits characters from 0 to 9, it defined as lowercase d
Syntax: - regex1= /\d/;
Example: -

```
let reg=/\d{1,}/g
console.log("RegExp:-",reg);
let str1="Give it 100%";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\D: - it matches not digits character not include from 0 to 9, it defined as uppercase D
Syntax: - regex1= /\d/;
Example: -

```
let reg=/\d{1,}/g
console.log("RegExp:-",reg);
let str1="Give it 100%";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\s: - it matches whitespace characters, it defined as lowercase s
Whitespace characters: -

- A space character ( )
- A tab character (\t)
- A carriage return character (\r)
- A new line character (\n)
- A vertical tab character (\v)
- A form feed character (\f)

Syntax: - regex1= /\s/;
Example: -

```
let reg=/\s/g
console.log("RegExp:-",reg);
let str1="Give it 100%";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
```

```
console.log("str1.matchAll(reg):-",res);
```

\S: - it matches non whitespace characters, it defined as uppercase S
Syntax: - regex1= /\S/;
Example: -

```
let reg=/\S{1,}/g
console.log("RegExp:-",reg);
let str1="Give it 100%";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\b: - it matches at beginning or end of a word
Syntax: - regex1= /\b(pattern)/;
Example1: - match beginning of word

```
let reg=/\b(he)/g
console.log("RegExp:-",reg);
let str1="he was here and i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

Syntax: - regex1= /(pattern)\b/;
Example2: - match end of word

```
let reg=/(re)\b/g
console.log("RegExp:-",reg);
let str1="he was here and i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\B: - it matches NOT at beginning or end of a word
Syntax: - regex1= /\B(pattern)/;
Example1: - match NOT beginning (end) of word

```
let reg=/\B(re)/g
console.log("RegExp:-",reg);
let str1="he was here and i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

Syntax: - regex1= /(pattern)\B/;
Example2: - match NOT end (begginning) of word

```
let reg=/(he)\B/g
console.log("RegExp:-",reg);
let str1="he was here and i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\0: - it matches the null character
Syntax: - regex1= /\0/;
Example: -

```
let reg=/\0/g
console.log("RegExp:-",reg);
let str1="he was here\0 and i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\n: - it matches newline character in string
Syntax: - regex1= /\n/;
Example: -

```
let reg=/\n/g
console.log("RegExp:-",reg);
let str1="he was here\ni was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\f: - it matches form feed character in string
Syntax: - regex1= /\f/;
Example: -

```
let reg=/\f/g
console.log("RegExp:-",reg);
let str1="he was here i was there\f";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\r: - it matches carriage return character in string
Syntax: - regex1= /\r/;
Example: -

```
let reg=/\r/g
console.log("RegExp:-",reg);
```

```
let str1="he was here\r i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\t: - it matches horizontal tab character in string
Syntax: - regex1= /\t/;
Example: -

```
let reg=/\t/g
console.log("RegExp:-",reg);
let str1="he was here\t i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

\v: - it matches vertical tab character in string
Syntax: - regex1= /\v/;
Example: -

```
let reg=/\v/g
console.log("RegExp:-",reg);
let str1="he was here\v i was there";
console.log("str1:-",str1);
let iter=str1.matchAll(reg);
let res=[...iter];
console.log("str1.matchAll(reg):-",res);
```

String validation: - RegExp are used for validation of string data

Mobile number validation: - match mobile number with starting 7,8,9 and 10 digit number
RegExp: - /^[789]\d{9}$/
Example: -

```
let vreg=/^[789]\d{9}$/;
console.log("Validation RegExp:-",vreg);
let data="8912345676";
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

Mobile number with +91 optional: - match mobile number with +91 is optional
RegExp: - /^(\+91)*[789]\d{9}$/
Example: -

```
let vreg=/^(\+91)*[789]\d{9}$/;
console.log("Validation RegExp:-",vreg);
let data="+919456778780";
```

```
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

Name only includes alphabets: -

RegExp: - /^[a-zA-Z]+$/

Example: -

```
let vreg=/^[a-zA-Z]+$/;
console.log("Validation RegExp:-",vreg);
let data="ram";
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

Password minimum 6 characters: - include special character, numbers and alphabets upper and lower case with minimum 6 characters length

RegExp: - /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&*_])[\w@$!%*?&*]{6,}$/

Example: -

```
let vreg=/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&*_])[\w@$!%*?&*]{6,}$/;
console.log("Validation RegExp:-",vreg);
let data="Ram@123";
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

Username starts with alphabets include number and underscore (_): -

RegExp: - /^[A-Za-z_]([a-zA-Z0-9_]){2,}$/

Example: -

```
let vreg=/^[A-Za-z_]([a-zA-Z0-9_]){2,}$/;
console.log("Validation RegExp:-",vreg);
let data="Ram123";
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

Email validation: -

RegExp: - /[a-zA-Z0-9_\-\.]+[@][a-z]+[\.][a-z]{2,3}$/

Example: -

```
let vreg=/[a-zA-Z0-9_\-\.]+[@][a-z]+[\.][a-z]{2,3}$/
console.log("Validation RegExp:-",vreg);
let data="Ram123@gmail.com";
console.log("Data:-",data);
let res=data.match(vreg);
console.log("Match:-",res);
```

NUMBERS: - it used for store number and manipute numbers using methods

Types of numbers: - javascript have two types of numbers as with decimals and without decimals, it can use scientific (exponent) notation for extra large or small number

Numbers with decimals: - the number can have decimal point
Example: -
```
let pi=3.14;
console.log("PI:-",pi);
```

Numbers withoutdecimals: - the number is without decimal point
Example: -
```
let r=3;
console.log("r:-",r);
```

Extra large numbers: - it uses exponent notation (e) for show extra large number
Syntax: - var1= (mantissa) e (exponent)
Example: -
```
let exp=4e5;
console.log("exp:-",exp);
let exp2=432e-5;
console.log("exp2:-",exp2);
```

Number precision: - accuracy of numbers in javascript

Integer precision: - integer numbers without exponent notation are accurate upto 15 digits
Example: -
```
let x = 999999999999999;   // x will be 999999999999999
console.log("x:-",x);
let y = 9999999999999999;  // y will be 10000000000000000
console.log("y:-",y);
```

Floating precision: - floating points in arithmetic is not always 100% accurate
Example: -
```
let x=0.2;
console.log("x:-",x);
let y=0.1;
console.log("y:-",y);
let z=x+y;
console.log("x+y:-",z);
```

Fix floating precision problem with multiply and divide: -
Example: -
```
let x=0.2;
console.log("x:-",x);
let y=0.1;
console.log("y:-",y);
```

```
let z=(x*10+y*10)/10;
console.log("x+y:-",z);
```

NaN – Not a Number: - it is reserved word for indicating that number is not legal number.

Value: - NaN
Syntax: - var1=NaN;
Example: -

```
let num=NaN;
console.log("Number:-",num);
```

Operations for NaN: - peform operation for see NaN value or operation with NaN value.
Example1: -

```
let x=10;
console.log("x:-",x);
let y="apple";
console.log("y:-",y);
let z=x/y;
console.log("x/y:-",z);
```

Example2: - operation with NaN value and number

```
let x=10;
console.log("x:-",x);
y=NaN;
console.log("y:-",y);
z=x+y;
console.log("x+y:-",z);
```

NaN type: - use typeof operator for find type of NaN value
Example: -

```
let res=typeof NaN; //number
console.log("typeof NaN:-",res);
```

isNaN: - it is method for check number is NaN or not
Example: -

```
let x=NaN;
console.log("x:-",x);
let res=isNaN(x);
console.log("isNaN(x):-",res);
```

Infinity: - Infinity or –Infinity is the javascript value will return if number is calculate outside the largest possible number.

Value: - Infinity
Syntax: - var1=Infinity;
Example: -

```
let x=Infinity;
console.log("x:-",x);
```

Operation for Infinity: -

Example1: - multiply number to infinity

```
let x=2;
while(x!=Infinity)
{
    x*=x;
    console.log(x);
}
```

Example2: - divide number by zero will give infinity

```
let x=10;
console.log("x:-",x);
let y=0;
console.log("y:-",y);
let z=x/y; //Infinity
console.log("x/y:-",z);
```

Example3: - divide negative number by zero will give negative infinity

```
let x=-10;
console.log("x:-",x);
let y=0;
console.log("y:-",y);
let z=x/y; //-Infinity
console.log("x/y:-",z);
```

Type of Infinity: - the typeof infinity value is number

Example: -

```
let res=typeof Infinity;
console.log("typeof Infinity:-",res);
```

Number Systems: - numbers of javascript can define with base10, base16, base8 and base2 number systems.

Decimal numbers: - this is default number system use with 0 to 9 numbers, it known as base10 number system.

Syntax: - var1=Number;

Example: -

```
let xd=10;
console.log("xd:-",x);
```

Hexadecimal numbers: - this number system use with 0 to 9 numbers and A to F alphabets, it known as base10 number system, it defined with 0x at starting of number

Syntax: - var1=0xNumber;

Example: -
```
let xh=0x10;
console.log("xh:-",xh);
```

Octal numbers: - this number system use with 0 to 7 numbers, it known as base8 number system, it defined with 0o at starting of number
Syntax: - var1=0oNumber;
Example: -
```
let xo=0o10;
console.log("xo:-",xo);
```

Binary numbers: - this number system use with 0 and 1 numbers, it known as base2 number system, it defined with 0b at starting of number
Syntax: - var1=0bNumber;
Example: -
```
let xb=0b10;
console.log("xb:-",xb);
```

Display numbers: - with toString() method output can get base2 to base36 numbers. This method returns base10 numbers by default.
Syntax: - num1.toString(radix arg); //radix args:- defualt 10, 2-36
Example: -
```
let x=13;
console.log("x:-",x);
let res1=x.toString();
console.log("x.toString():-",res1);
let res2=x.toString(16);
console.log("x.toString(16):-",res2);
let res3=x.toString(2);
console.log("x.toString(2):-",res3);
```

Number object: - it can also define using new key word before Number function, but it slow down execution speed.

Syntax: - var1=new Number(value);
Example: -
```
let x=new Number(13);
console.log("x:-",x);
```

Typeof number: -
Example: -
```
let x=new Number(13);
console.log("x:-",x);
let res= typeof x;
console.log("typeof x:-",res);
```

Comparison of numbers: - comparison of number literal and number object
Example: -
```
let x=new Number(13);
console.log("x:-",x);
let y=Number(13);
console.log("y:-",y);

let res1=x==y; //value comparison
console.log("x==y:-",res1); //true
let res2=x===y; //identical comparison
console.log("x===y:-",res2); //false
```

BigInt: - these variables are used to store big integer values that are too big for represented by normal Numbers variables. It is another datatype of javascript

Integer Accuracy: - javascript integers are accurate only upto 15 digits, it store all numbers in 64-bit floating point format. With this format large integer cannot be exactly represent and it will be rounded, it can only safely represent integers.
Up to 9007199254740991 +(253-1)
Down to -9007199254740991 -(253-1).
Index outside of this range can lose accuracy.

Syntax: - it can define with BigInt function or append n to end of integer number.
```
Var1=BigInt(number);
Var1=valn;
```
Example: -
```
let x=BigInt(12345678901234567890123456789012345);
console.log("x:-",x);
let y=12345678901234567890123456789012345n;
console.log("y:-",y);
```

Typeof bigint: - check type of bigint using typeof operator
Exmple: -
```
let x=25n;
console.log("x:-",x);
let res=typeof x;
console.log("typeof x:-",res);
```

BigInt decimals: - it does not have decimals points, big int supports integers only, it will throw error if operations with bigint results decimals points values
Example: -
```
let x=5n/2; //throw TypeError
console.log("5n/2:-",x);
```

Bigint hex, octal and binary numbers: - hex, octal and binary numbers can also define with bigint data type.

Hexadecimal number: - let x=0x10n;
Ocatal number: - let x=0o10n;
Binary number: - let x=0b10n;

Example: -
```
let hex = 0x20000000000003n;
console.log("hex:-",hex);
let oct = 0o400000000000000003n;
console.log("oct:-",oct);
let bin = 0b10000000000000000000000000000000000000000000000000000011n;
console.log("bin:-",bin);
```

Number properties: -

Number.EPSILON: - this data property represents the difference between 1 and the smallest floating point number greater than 1.
Syntax: - var1=Number.EPSILON;
Example: -
```
let x=Number.EPSILON;
console.log("Number.EPSILON:-",x);
```

Number.MAX_VALUE: - it constant representing the largest possible number in JavaScript
Syntax: - var1=Number.MAX_VALUE;
Example: -
```
let x=Number.MAX_VALUE;
console.log("Number.MAX_VALUE:-",x);
```

Number.MIN_VALUE: - it constant representing the smallest possible number in JavaScript
Syntax: - var1=Number.MIN_VALUE;
Example: -
```
let x=Number.MIN_VALUE;
console.log("Number.MIN_VALUE:-",x);
```

Number.MAX_SAFE_INTEGER: - it representing the maximum safe integer number in JavaScript
Syntax: - var1=Number.MAX_SAFE_INTEGER;
Example: -
```
let x=Number.MAX_SAFE_INTEGER;
console.log("Number.MAX_SAFE_INTEGER:-",x);
```

Number.MIN_SAFE_INTEGER: - it representing the minimum safe integer number in JavaScript
Syntax: - var1=Number.MIN_SAFE_INTEGER;
Example: -
```
let x=Number.MIN_SAFE_INTEGER;
```

```
console.log("Number.MIN_SAFE_INTEGER:-",x);
```

Number.POSITIVE_INFINITY: - it returns positive infinity
Syntax: - var1=Number.POSITIVE_INFINITY;
Example: -

```
let x=Number.POSITIVE_INFINITY;
console.log("Number.POSITIVE_INFINITY:-",x);
```

Number.NEGATIVE_INFINITY: - it returns negative infinity
Syntax: - var1=Number.NEGATIVE_INFINITY;
Example: -

```
let x=Number.NEGATIVE_INFINITY;
console.log("Number.NEGATIVE_INFINITY:-",x);
```

Number.NaN: - it is reserved word for number that is not legal number
Syntax: - var1=Number.NaN;
Example: -

```
let x=Number.NaN;
console.log("Number.NaN:-",x);
```

Number methods: -

toString: - this method return numbers as string
Syntax: - var1=num1.toString();
Example: -

```
let x=20;
console.log(x);
let str1=x.toString();
console.log(str1);
```

toExponential: - this method return string with rounded number, written using exponential notation. It also take optional parameter for defines number of character behind the decimal point
Syntax: - var1=num1.toExponential(value);
Exmaple1: -

```
let x=9.345;
console.log("x:-",x);
let str1=x.toExponential(1);
console.log("x.toExponential(1):-",str1);
let str2=x.toExponential();
console.log("x.toExponential():-",str2);
```

Example2: -

```
let x=50000;
console.log("x:-",x);
let str1=x.toExponential(2);
console.log("x.toExponential(2):-",str1);
```

```
let str2=x.toExponential();
console.log("x.toExponential():-",str2);
```

Example3: -

```
let x=0.00025;
console.log("x:-",x);
let str1=x.toExponential(2);
console.log("x.toExponential(2):-",str1);
let str2=x.toExponential();
console.log("x.toExponential():-",str2);
```

toFixed: - it return number as string with specified decimal numbers pass as paramter, it remove all decimal number if no parameter passed.
Syntax: - var1=num1.toFixed(value);
Example: -

```
let x=9.3759;
console.log("x:-",x);
let str1=x.toFixed(3);
console.log("x.toFixed(3):-",str1);
let str2=x.toFixed(2);
console.log("x.toFixed(2):-",str2);
let str3=x.toFixed(0);
console.log("x.toFixed(0):-",str3);
let str4=x.toFixed();
console.log("x.toFixed():-",str4);
let str5=x.toFixed(6);
console.log("x.toFixed(6):-",str5);
```

toPrecision: - this method return number as string, with specified length that passed as parameter
Syntax: - var1=num1.toPrecision(length);
Example: -

```
let x=23.3478;
console.log("x:-",x);
let str1=x.toPrecision(1);
console.log("x.toPrecision(1):-",str1);
let str2=x.toPrecision(2);
console.log("x.toPrecision(2):-",str2);
let str3=x.toPrecision(3);
console.log("x.toPrecision(3):-",str3);
str4=x.toPrecision(4);
console.log("x.toPrecision(4):-",str4);
```

valueOf: - this method return number as number
Syntax: - var1=num1.valueOf();
Example: -

```
let x=45;
```

```
console.log("x:-",x);
let res=x.valueOf();
console.log("res:-",res);
```

Number: - it returns number converted from argument
Syntax: - var1=Number(value);
Example: -

```
let x=Number("5.34");
console.log("x:-",x);
```

Number.parseInt: - it parse string and return integer, spaces are allowed, and only first number will return
Syntax1: - var1=Number.parseInt("string"); or var1=parseInt("string");
Example1: -

```
let x=Number.parseInt("37%");
console.log("x:-",x);
```

Number.parseFloat: - it parse string and return flaoting point number, spaces are allowed, and only first number will return
Syntax: - var1=Number.parseFloat("string"); or var1=parseFloat("string");
Example: -

```
let x=Number.parseFloat("3.56a");
console.log("x:-",x);
```

Number.isInteger: - it return true if argument is integer
Syntax: - var1=Number.isInteger(value);
Example: -

```
let x=5;
console.log("x:-",x);
let bool=Number.isInteger(x);
console.log("Number.isInteger(x):-",bool);
```

Number.isSafeInteger: - it return true if argument is safe integer. Safe integer can be exactly represented as a double precision number.
Safe integers are all integers from -(253 - 1) to +(253 - 1).
Syntax: - var1=Number.isInteger(Number);
Example: -

```
let x=5421423423646456;
console.log("x:-",x);
let bool=Number.isSafeInteger(x);
console.log("Number.isSafeInteger(x):-",bool);
```

Number.isFinite: - this method return true is number is finite. If infinite numbers occures as Infinity, -Infinity or NaN, it will return false.
Syntax: - var1=Number.isFinite(Number);
Example: -

```
let x=123;
console.log("x:-",x);
let res=Number.isFinite(x);
console.log("Number.isFinite(x):-",res);
```

Number.isNaN: - this method return true if number value is NaN.
Syntax: - var1=Number.isNaN(Number);
Example: -

```
let x=Number("good");
console.log("x:-",x);
let res=Number.isNaN(x);
console.log("Number.isNaN(x):-",res);
```

Global Number methods: -

isFinite: - it return true if number is finite. This global method converts value to number before using it.
Syntax: - var1=isFinite(value);
Example: -

```
let x=isFinite("123");
console.log("isFinite():- ",x); //true

let y=Number.isFinite("123");
console.log("Number.isFinite():- ",y); //false
```

isNaN: - it return true if number is NaN. This global method converts value to number before using it.
Syntax: - var1=isNaN(value);
Example: -

```
let x=isNaN("Good");
console.log("isNaN():- ",x); //true

let y=Number.isNaN("Good");
console.log("Number.isNaN():- ",y); //false
```

parseInt: - it return first number as integer. It is global method can use without number object
Syntax: - var1=parseInt(value);
Example: -

```
let x=parseInt("1 day");
console.log("parseInt():- ",x);
```

parseFloat: - it return first number as float. It is global method can use without number object
Syntax: - var1=parseFloat(value);
Example: -

```
let x=parseFloat("1 day");
console.log("parseFloat():- ",x);
```

MATH: - Math object is used for perform mathematical task. It is static object, and does not have constructor, Math object's method and properties can be used without creating Math object.

Math properties: - javascript provides 8 mathematical constants that can be access with math properties.

Math.E: - this property returns Euler's number
Euler's number: - it is an important constant that is found in many contexts and is the base for natural logarithms.
Syntax: - var1=Math.E;
Example: -
```
let x=Math.E;
console.log("Math.E:-",x);
```

Math.PI: - this property returns value of PI, use for find circumference or area of circle.
Syntax: - var1=Math.PI;
Example: -
```
let x=Math.PI;
console.log("Math.PI:-",x);
```

Math.SQRT2: - this property returns the square root of 2
Syntax: - var1=Math.SQRT2;
Example: -
```
let x=Math.SQRT2;
console.log("Math.SQRT2:-",x);
```

Math.SQRT1_2: - this property returns the square root of 1/2
Syntax: - var1=Math.SQRT1_2;
Example: -
```
let x=Math.SQRT1_2;
console.log("Math.SQRT1_2:-",x);
```

Math.LN2: - this property returns natural logarithm of 2
Syntax: - var1=Math.LN2;
Example: -
```
let x=Math.LN2;
console.log("Math.LN2:-",x);
```

Math.LN10: - this property returns natural logarithm of 10
Syntax: - var1=Math.LN10;
Example: -
```
let x=Math.LN10;
console.log("Math.LN10:-",x);
```

Math.LOG2E: - this property returns base 2 logarithm of E (Euler's number)
Syntax: - var1=Math.LOG2E;
Example: -
```
let x=Math.LOG2E;
console.log("Math.LOG2E:-",x);
```

Math.LOG10E: - this property returns base 10 logarithm of E (Euler's number)
Syntax: - var1=Math.LOG10E;
Example: -
```
let x=Math.LOG10E;
console.log("Math.LOG10E:-",x);
```

Math methods: -

Math.round: - it returns nearest integer number
Syntax: - var1=Math.round(number);
Example1: -
```
let v=4.6;
console.log("v:-",v);
let x=Math.round(v);
console.log("Math.round(v):-",x);
```

Example2: -
```
let v=4.4;
console.log("v:-",v);
let x=Math.round(v);
console.log("Math.round(v):-",x);
```

Math.ceil: - it returns value rounded up to its nearest integer
Syntax: - var1=Math.ceil(number);
Example: -
```
let v=4.3;
console.log("v:-",v);
let x=Math.ceil(v);
console.log("Math.ceil(v):-",x);
```

Math.floor: - it returns value rounded down to its nearest integer
Syntax: - var1=Math.floor(number);
Example: -
```
let v=4.7;
console.log("v:-",v);
let x=Math.floor(v);
console.log("Math.floor(v):-",x);
```

Math.trunc: - it returns integer part of value.
Syntax: - var1=Math.trunc(number);

Example: -

```
let v=4.9;
console.log("v:-",v);
let x=Math.trunc(v);
console.log("Math.trunc(v):-",x);
```

Math.sign: - it returns 1, 0 and -1 if value is respectively positive, null or zero and –negative.
Syntax: - var1=Math.sign(number);
Example1: -

```
let v=null;
console.log("v:-",v);
let x=Math.sign(v);
console.log("Math.sign(v):-",x);
```

Example2: -

```
let v=5.6;
console.log("v:-",v);
let x=Math.sign(v);
console.log("Math.sign(v):-",x);
```

Example3: -

```
let v=-6.7;
console.log("v:-",v);
let x=Math.sign(v);
console.log("Math.sign(v):-",x);
```

Math.pow: - it returns value of number to times of power.
Syntax: - var1=Math.pow(number,power);
Explain: - var1=Math.pow(n1,3); //n1*n1*n1;
Example: -

```
let v=5;
let r=3;
console.log("v:-",v,"r:-",r);
let x=Math.pow(v,r);
console.log("Math.pow(v,r):-",x);
```

Math.sqrt: - it returns square root of value.
Syntax: - var1=Math.sqrt(number);
Example: -

```
let v=5;
console.log("v:-",v);
let x=Math.sqrt(v);
console.log("Math.sqrt(v):-",x);
```

Math.abs: - it returns absolute (positive) value
Syntax: - var1=Math.abs(number);

Example: -

```
let v=-5;
console.log("v:-",v);
let x=Math.abs(v);
console.log("Math.abs(v):-",x);
```

Math.min: - it returns lowest or minimum value from argument list

Syntax: - var1=Math.min(arg1,arg2,…,argN);

Example1: -

```
let x=Math.min(23,1,35,5,68);
console.log("Math.min(23,1,35,5,68):-",x);
```

Example2: -

```
let x=Math.min(3,4,6);
console.log("Math.min(3,4,6):-",x);
```

Math.max: - it returns lowest or maximum value from argument list

Syntax: - var1=Math.max(arg1,arg2,…,argN);

Example1: -

```
let x=Math.max(23,1,35,5,68);
console.log("Math.max(23,1,35,5,68):-",x);
```

Example2: -

```
let x=Math.max(3,4,6);
console.log("Math.max(3,4,6):-",x);
```

Math.random: - it returns random numbers between 0 (inclusive) and 1(exclusive)

Syntax: - var1=Math.random();

Example: -

```
let x=Math.random();
console.log("Math.random():-",x);
```

Math.log: - it returns the natural logarithm of value. It finds number that how many times multiplies Euler's number(Math.E) to get value.

Syntax: - var1=Math.log(number);

Example: -

```
let v=15;
console.log("v:-",v);
let x=Math.log(v);
console.log("Math.log(v):-",x);
```

Verification: -

```
let vv=Math.pow(Math.E,x);
console.log(vv);
```

Math.log2: - it returns the base 2 logarithm of value. It finds number that how many times multiplies 2 to get value.
Syntax: - var1=Math.log2(number);
Example: -

```
let v=15;
console.log("v:-",v);
let x=Math.log2(v);
console.log("Math.log2(v):-",x);
```

Verification: -

```
let vv=Math.pow(2,x);
console.log(vv);
```

Math.log10: - it returns the base 10 logarithm of value. It finds number that how many times multiplies 10 to get value.
Syntax: - var1=Math.log10(number);
Example: -

```
let v=15;
console.log("v:-",v);
let x=Math.log10(v);
console.log("Math.log10(v):-",x);
```

Verification: -

```
let vv=Math.pow(10,x);
console.log(vv);
```

Math.sin: - it returns sine value (between -1 and 1) of given angle in radian
Syntax: - var1=Math.sin(radian);
Example: -

```
let v=1.5707;
console.log("v:-",v);
let x=Math.sin(v);
console.log("Math.sin(v):-",x);
```

Convert angle degree to radian: -

Formula: - radian=angle in degree * Math.PI /180
Example: -

```
let angle=90;
console.log("angle:-",angle);
let radian=angle*Math.PI/180;
console.log("radian:-",radian);
```

Math.cos: - it returns cosine value (between -1 and 1) of given angle in radian
Syntax: - var1=Math.sin(radian);
Example: -

```
let v=1.5707;
```

```
console.log("v:-",v);
let x=Math.sin(v);
console.log("Math.sin(v):-",x);
```

Math.cosh: - this method returns the hyperbolic cosine of number
Syntax: - var1=Math.cosh(number);
Example: -

```
let v=3;
console.log("v:-",v);
let x=Math.cosh(v);
console.log("Math.cosh(v):-",x);
```

Math.acos: - this method return arccosine (in randians) of value, it is between 0 to Math.PI value. It expect in value in range of -1 and 1.
Syntax: - var1=Math.acos(number);
Example: -

```
let v=0.4;
console.log("v:-",v);
let x=Math.acos(v);
console.log("Math.acos(v):-",x);
```

Math.acosh: - this method return the hyperbolic arc-cosine (in randians) of value, it returns NaN if value is less than 1
Syntax: - var1=Math.acosh(number);
Example: -

```
let v=10;
console.log("v:-",v);
let x=Math.acosh(v);
console.log("Math.acosh(v):-",x);
```

Math.asin: - this method return arcsine (in randians) of value, it is between –PI/2 to PI/2 value. It expect in value in range of -1 and 1.
Syntax: - var1=Math.asin(number);
Example: -

```
let v=0.4;
console.log("v:-",v);
let x=Math.asin(v);
console.log("Math.asin(v):-",x);
```

Math.asinh: - this method returns the hyperbolic arc-sine (in randians) of value
Syntax: - var1=Math.asinh(number);
Example: -

```
let v=10;
console.log("v:-",v);
let x=Math.asinh(v);
console.log("Math.asinh(v):-",x);
```

Math.sinh: - this method returns the hyperbolic sine of value
Syntax: - var1=Math.sinh(number);
Example: -

```
let v=2;
console.log("v:-",v);
let x=Math.sinh(v);
console.log("Math.sinh(v):-",x);
```

Math.tan: - this method return tangent of value, it argument value as radian
Syntax: - var1=Math.tan(number);
Example: -

```
let v=1;
console.log("v:-",v);
let x=Math.tan(v);
console.log("Math.tan(v):-",x);
```

Math.tanh: - this method returns the hyperbolic tangent of value
Syntax: - var1=Math.tanh(number);
Example: -

```
let v=2;
console.log("v:-",v);
let x=Math.tanh(v);
console.log("Math.tanh(v):-",x);
```

Math.atan: - this method returns arctangent of value between –PI/2 and PI/2 radians.
Syntax: - var1=Math.atan(number);
Example: -

```
let v=1;
console.log("v:-",v);
let x=Math.atan(v);
console.log("Math.atan(v):-",x);
```

Math.atan2: - this method returns the arctangent of the quotient of its arguments, as a numeric value between PI and -PI radians.
The number returned represents the counterclockwise angle in radians (not degrees) between the positive X axis and the point (x, y).
Note: - With atan2(), the y coordinate is passed as the first argument and the x coordinate is passed as the second argument.
Syntax: - var1=Math.atan2(y-coordinates, x-coordinates);
Example: -

```
let xc=4;
let yc=8;
console.log("xc:-",xc,"yc:-",yc);
let x=Math.atan2(yc,xc);
console.log("Math.atan2(yc,xc):-",x);
```

Convert angle radian to degree: -
Formula: - angle in degree=radian* 180/Math.PI
Example: -

```
let radian=1.1071;
console.log("radian:-",radian);
angle=radian*180/Math.PI;
console.log("angle:-",angle);
```

Math.atanh: - this method returns the hyperbolic arctangent (in randians) of value. It expect in value in range of -1 and 1. It returns Infinity if value is 1 and –Infinity if value is -1.
Syntax: - var1=Math.atanh(number);
Example: -

```
let v=10;
console.log("v:-",v);
let x=Math.atanh(v);
console.log("Math.atanh(v):-",x);
```

Math.cbrt: - this method returns cubic root of number.
Syntax: - var1=Math.cbrt(number);
Example: -

```
let v=27;
console.log("v:-",v);
let x=Math.cbrt(v);
console.log("Math.cbrt(v):-",x);
```

Math.clz32: - this method return value as count of leading zero in 32-bit binary number
Syntax: - var1=Math.clz32(number);
Example1: -

```
let v=2;
console.log("v:-",v);
9494949494949494949494949494949494949494
console.log("Math.clz32(v):-",x);
```

Example2: -

```
let v=4;
console.log("v:-",v);
let x=Math.clz32(v);
console.log("Math.clz32(v):-",x);
```

Example3: -

```
let v=6;
console.log("v:-",v);
let x=Math.clz32(v);
console.log("Math.clz32(v):-",x);
```

Math.hypot: - it returns hypotenious of triangle by given two sides
Syntax: - var1=Math.hypot(side1,side2);
Example: -

```
let a=3;
let b=4;
console.log("a:-",a,"b:-",b);
let x=Math.hypot(a,b);
console.log("Math.hypot(a,b):-",x);
```

Math.exp: - this method returns E^value of given value, where E is Eular's number (Math.E)
Syntax: - var1=Math.exp(number);
Example: -

```
let v=3;
console.log("v:-",v);
let x=Math.exp(v);
console.log("Math.exp(v):-",x);
```

Math.expm1: - this method returns E^value minus 1 of given value, where E is Eular's number (Math.E). This method is more accurate than Math.exp() and subtracting 1
Syntax: - var1=Math.expm1(number);
Example: -

```
let v=3;
console.log("v:-",v);
let x=Math.expm1(v);
console.log("Math.expm1(v):-",x);
```

Math.fround: - this method returns nearest 32-bit single presion float representation of number
Syntax: - var1=Math.fround(number);
Example: -

```
let v=3.7;
console.log("v:-",v);
let x=Math.fround(v);
console.log("Math.fround(v):-",x);
```

Math.random: - use Math.random method

Math.random method: - it returns random numbers between 0 (inclusive) and 1(exclusive)
Syntax: - var1=Math.random();
Example: -

```
let x=Math.random();
console.log("Math.random():-",x);
```

Get integer from 0 to 10 range: -
Example: -

```
let x=Math.floor(Math.random()*11);
console.log("Random in range 0 to 10:-",x);
```

Get integer from 0 to 100 range: -

Example: -

```
let x=Math.floor(Math.random()*101);
console.log("Random in range 0 to 101:-",x);
```

Get integer from 1 to 10 range: -

Example: -

```
let x=Math.floor(Math.random()*10)+1;
console.log("Random in range 1 to 10:-",x);
```

Get integer from 1 to 100 range: -

Example: -

```
let x=Math.floor(Math.random()*100)+1;
console.log("Random in range 1 to 100:-",x);
```

Random function for take values in given range: -

Example: -

```
function getRandomInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
}

let x=getRandomInteger(1,10);
console.log("getRandomInteger(1,10):-",x);
```

STRICT MODE: - write JavaScript in strict mode, for minimum runtime or logical errors, Strict mode makes it easier to write "secure" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors.

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.

Declaration: - it is declare by adding "use strict"; to the beginning of script or functions

Example: -

Beginning of script: -

```
<script type="text/javascript">
    "use strict";
    …
</script>
```

Beginning of function: -

```
function good(){
    "use strict";
    …
```

}

Note: - "use strict"; directive is only recognized at the beginning of script or function

Not allowed in strict mode: - syntax that not allowed in strict mode

Using variable or objects without declaration: - JavaScript define undeclared variables/objects as global variables/objects, but it strict mode it will throw error

Example: -

```
v=34; //this will throw error for undeclared variable
console.log("v:-",v);
```

Deleting variable or objects: - it is restricted in strict mode

Example: -

```
let x=45;
console.log("x:-",x);
delete x; //deleting variable will throw error
```

Deleting functions: - it is also restricted in strict mode

Example: -

```
function good(){
    ...
}

delete good; //deleting function will throw error
```

Duplicate parameters in function: - duplicate parameters are not allowed in strict mode

Example: -

```
function y(a1,a1){} //this will throw error
```

Octal numeric literals are not allowed: -

Example: -

```
let x = 010; //this will throw error
let a=0o10; //define octal using o after zero.
console.log(a);
```

Octal escape characters are not allowed: -

Example: -

```
let x= '\010'; //this will throw error
```

Writing read-only property is not allowed: -

Example: -

```
const obj = {};
Object.defineProperty(obj, "x", {value:10, writable:false});
obj.x = 3.14; //this will throw error
```

Writing get-only property is not allowed: -

Example: -

```
const obj = {get x() {return 0} };
obj.x = 3.14; //this will throw error
```

Deleting an undeletable property is not allowed: -

Example: -

```
delete Object.prototype; //this will throw error
```

Word eval cannot use as variable: -

Example: -

```
let eval = 3.14; //this will throw error
```

Word arguments cannot use as variable: -

Example: -

```
let arguments = 3.14;  //this will throw error
```

With statement is not allowed: -

Example: -

```
with (Math){x = max(34,45,67)}; //this will throw error
console.log(x);
```

Note: - with statement is **deprecated**

eval is not allowed to create global variables and use them outside eval: -

Example: -

```
eval("x = 2");
console.log(x);

eval("var z = 6");
console.log(x);
```

example2: - also cannot use block scope variables outside eval function

```
eval("let d = 4;console.log(d);");
console.log(d);
```

this keyword return undefined instead of window object: -

Example: -

```
function func1(){
    console.log(this);
}
func1();
```

Future proof: - keywords reserved for future JavaScript versions not allowed to used as variable names

Keywords: -

- implements

- interface
- let
- package
- private
- protected
- public
- static
- yield

Example: -
let implements=45;

HOISTING: - it is javascript feature for access variables and function before declaration.

Variables: - variables with var can access (use) before declare them, but let and const block scope variables cannot used before declare.

Var: - using var variables before declaring them
Example: -
"use strict"
v=34;
document.write("v:- ",v);
var v;

let: - let variables cannot use before declaring them, it will throw RefernceError for cannot access 'variable' before initialization.
Example: -
l=34; //this will throw error
document.write("l:- ",l);
let l;

const: - const variables cannot use before declaring them, it will throw SyntaxError for missing initializer in const declaration.
Example: -
c=34; //this will throw error
document.write("c:- ",c);
const c;

Functions: - functions can also called or invoke before defination of function.

Syntax: -
Function call();
Function defination
…
…

Example: -

```
greet(45); //function call

function greet(name)  //function defination
{
    document.write("Good day ",name,"<br>");
}
```

OBJECTS2: -

Arguments object: - this object contains an array of arguments used when the function is invoked.

Syntax: -

```
function function_name()
{
    arguments
}
```

Example: -

```
function add()
{
    sum=0;
    for(let i=0;i<arguments.length;i++)
    {
        sum+=arguments[i];
    }
    console.log("add:- ",sum);
}

add(34,45,6,77,88,89,90,00);
add(1,2,3,4,5,6,7,8,9,10);
```

Object accessor: - get and set keywords are used for access or assign values to properties of object. This accessor are used with function defined.

Set: - it is used for defined function with argument for set property value.

Syntax: -

```
Obj1={
    property1:value1,
    set function_name(param1)
    {
        this.property1=param1;
    }
```

```
}
```

Example: -

```
let person={
    name: "ram",
    set setname(n){
        if(typeof n=="string"){
            this.name=n;
        }
    }
}
console.log(person);
person.setname="sham";
console.log(person);
```

Get: - it is used for defined function with argument for get property value.

Syntax: -

```
Obj1={
    property1:value1,
    get function_name()
    {
        return this.property1;
    }
}
```

Example: -

```
let person={
    name: "ram",
    get getname(){
        return this.name;
    }
}
console.log(person);
console.log("Name: -",person.getname);
```

get and set example: -

```
let person={
    name: "ram",
    set setname(n){
        if(typeof n=="string"){
            this.name=n;
        }
    },
    get getname(){
        return this.name;
    }
}
```

```
console.log(person);
person.setname="sham";
console.log(person.getname);
console.log(person);
```

why use getter and setter?: -

- It gives simpler syntax
- It allows equal syntax for properties and methods
- It can secure better data quality
- It is useful for doing things behind-the-scenes

Object.defineProperty: - this method can also used for add getters and setters.

Example: -

```
// Define object
const obj = {counter : 3};

// Define setters and getters
Object.defineProperty(obj, "reset", {
  get : function () {this.counter = 0;}
});
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
Object.defineProperty(obj, "subtract", {
  set : function (value) {this.counter -= value;}
});

// Play with the counter:
console.log("obj:-",obj);
obj.reset;
console.log("obj (reset):-",obj);
obj.add = 5;
console.log("obj (add=5):-",obj);
obj.subtract = 1;
console.log("obj (subtract=1):-",obj);
obj.increment;
console.log("obj (increment):-",obj);
obj.decrement;
console.log("obj (decrement):-",obj);
```

Object constructor functions: - this function used for create many objects with same properties and method. It creates object type for create multiple objects.

Syntax: -

```
function Object_type_name(param1,param2, …, paramN){
    this.property1=param1;
    this.property2=param2;
    …
    this.propertyN=paramN;
}

Obj_name= new Object_type(arg1, arg2, …, argN);
```

Example: -

```
function Person(first, age, eye) {
    this.firstName = first;
    this.age = age;
    this.eyeColor = eye;
}

const p1 = new Person("ram", 50, "blue");
console.log(p1);

const p2 = new Person("sham", 48, "green");
console.log(p2);

const p3 = new Person("madhav", 18, "green");
console.log(p3);
```

Adding property to constructor: - add new property to function constructor using prototype.

Syntax: -

```
Object_type.prototype.new_property=value;
```

Example: -

```
Person.prototype.email=undefined;

const p3 = new Person("sham", 48, "green");
console.log(p3);
p3.email="sham48@gmail.com";
console.log(p3.firstName,":-",p3.email);
```

Method in constructor function: -

Syntax: -

```
function Object_type_name(param1,param2){
    this.property1=param1;
    this.property2=param2;
    this.method1=function(){
```

```
        //block of code
    }
}

Obj_name= new Object_type(arg1, arg2);
Obj_name.method1();
```

Example: -

```
function Person(first, age, eye) {
    this.firstName = first;
    this.age = age;
    this.username = function(){
        return this.firstName+this.age;
    }
}

const p2 = new Person("sham", 48, "green");
console.log(p2);
console.log("Username:-",p2.username());
```

Adding method to constructor: - add new method to function constructor using prototype.

Syntax: -

```
Object_type.prototype.new_method=function(){
    //block of code
}
```

Example: -

```
Person.prototype.changeName = function (name) {
    this.firstName = name;
}

const p4 = new Person("madhav", 48, "green");
console.log(p3);
p3.changeName("krishna");
console.log(p3);
```

Sets object: - it is collection of unique values. Each value can only occur once in a Set. The values can be of any type, primitive values or objects.

Syntax: - var1=new Set(array of values);

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
console.log(s);
```

Typeof sets: - check type and instance of set

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let type=typeof s;
console.log("typeof s:- ",type);
let instance= s instanceof Set;
console.log("s instanceof Set:- ",instance);
```

Iterating sets: - use for..of loop for iterating set

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
for(let x of s)
{
    console.log(x);
}
```

Size property: - this property returns the number of elements in Set.

Syntax: - var1=set1.size

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let length=s.size;
console.log("count of elements in set:-",length);
```

Methods of set: -

add: - this method adds new elements to array

Syntax: - set1.add(value);

Example: - add elements to empty set

```
let s2=new Set();
s2.add(34);
s2.add(3);
s2.add(5);
s2.add(3);
console.log(s2);
```

has: - this returns true if set have specified value.

Syntax: - var1=set1.has(value1);

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let r=s.has(45);
console.log("s.has(45):-",r);
```

forEach: - it invoke passed function for every element in set, that passed function take element value as parameter.

Syntax: -

```
set1.forEach(function(value of set){
    //block of code
});
```

Example: -

```
let sum=0;
s.forEach(function(value){
    sum+=value;
});
console.log("sum:-",sum);
```

values: - this method returns an iterator object with the values in a set.
Syntax: - var1=set1.values();
Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let iter=s.values();
console.log(iter);
```

keys: - this method returns an iterator object with the values in a set, set object does not have keys it will work same as values method.
Syntax: - var1=set1.keys();
Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let iter=s.keys();
console.log(iter);
for(let x of iter)
{
    console.log(x);
}
```

iterator: - iterators return by keys and values method, these iterator have next method for print object of value.
Example: -

```
let iter=s.keys();
console.log(iter.next());
```

entries: - this method returns iterator with [value, value] pairs from set, entries method return key and value pair but set object does not have keys it returns [value,value] pair.
Syntax: - var1=s.entries();
Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let pairs=s.entries();
for(let x of pairs)
{
   console.log(x);
}
```

delete: - this method remove specified value from a set, and return true of value existed otherwise false.
Syntax: - var1=set1.delete(value);
Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
let r=s.delete(45);
console.log(s,"result:-",r);
```

clear: - this method remove all values from a set

Syntax: - set1.clear();

Example: -

```
let s=new Set([3,45,7,6,7,3,4]);
s.clear();
console.log(s);
```

Map object: - A Map holds key-value pairs where the keys can be any datatype. A Map remembers the original insertion order of the keys.

Syntax: -

var1=new Map(array of key and value pairs);

Example: -

```
let m=new Map([
    ["apples",344],
    ["bananas",234],
    ["oranges",123]
]);
console.log(m);
```

Typeof maps: - check type and instance of map

Example: -

```
let m=new Map([
    ["apples",344],
    ["bananas",234],
    ["oranges",123]
]);
let type=typeof m;
console.log("typeof s:- ",type);
let instance= m instanceof Map;
console.log("m instanceof Map:- ",instance);
```

Iterating maps: - use for..of loop for iterating map

Example: -

```
let m=new Map([
    ["apples",344],
    ["bananas",234],
    ["oranges",123]
]);
for(let x of m)
{
```

```
    console.log(x,x[0],x[1]);
}
```

Size property: - this property returns the number of elements in Set.
Syntax: - var1=map1.size
Example: -

```
let m=new Map([
    ["apples",344],
    ["bananas",234],
    ["oranges",123]
]);
let length=m.size;
console.log("count of key values pairs in maps:-",length);
```

Methods of map: -

set: - it add elements in map or change exisiting map values.
Syntax: - map1.set("key",value);
Example: -

```
m.set("mangoes",312);
console.log(m);
```

get: - it return value of given key from map
Syntax: - var1=map1.get("key");
Example: -

```
let x=m.get("apples");
console.log("apples:-",x);
```

delete: - this method removes elements from map of given key and return true if it find key
Syntax: - var1=map1.delete("key");
Example: -

```
let x=m.delete("bananas");
console.log(m,x);
```

has: - this method return true if map have element of given key
Syntax: - var1=map.has("key");
Example: -

```
x=m.has("oranges");
console.log("has oranges:- ",x);
```

forEach: - this method invoke callback function for every element of map, this callback function get two parameters as value and key
Syntax: -

```
map1.forEach(function(value,key){
    // block of code
});
```

Example: -

```
m.forEach(function(value,key){
    console.log(key,"=>",value);
});
```

entries: - this method returns an iterator object with the [key,values] in a map
Syntax: - var1=map1.entries();
Example: -

```
let iter=m.entries();
console.log(iter);

for(let x of iter)
{
    console.log(x);
}
```

keys: - this method returns an iterator object with the keys in a map
Syntax: - var1=map1.keys();
Example: -

```
iter=m.keys();
console.log(iter);
```

values: - this method returns an iterator object with the values in a map
Syntax: - var1=map1.values();
Example: -

```
iter=m.values();
console.log(iter);
```

clear: - this method remove all elements from map
Syntax: - map1.clear();
Example: -

```
m.clear();
console.log(m);
```

Date object: - this object work with dates and times, JavaScript will use the browser's time zone and display a date as a full text string.

Create date object: -

Method1: - it create date object with current date and time.
Syntax1: - var1= new Date();
Example: -

```
const date=new Date();
console.log(date);
```

Method2: - it create date object with string arguments with specified formats.
Syntax1: - var1= new Date("Year-month-date(number format)");
Example: -

```
const date=new Date("2024-3-14");
```

```
console.log(date);
```

Syntax2: - var1= new Date("Month(name) date, year hours:minute:seconds");
Example: -
```
date2=new Date("July 24, 2020 11:13:00");
console.log(date2);
```

Method3: - it take multiple arguments in numbers for year, month, day, hour, minute, seconds and milliseconds. Adding greater values than given range will overflow in previous value.
Values range: -
Year: - negative or positive values.Ex: - -100, 2028
Month: - 0 to 11 respectively january to december. Ex: - 2(March), 4(May)
Day: - 1 to 31 according to months. Ex: - 3, 7
Hour: - 0 to 23. Ex: - 5, 6
Minute: - 0 to 59. Ex: - 45,57
Seconds: - 0 to 59. Ex: - 30,40
Milliseonds: - 0 to 999. Ex: - 300, 700

Syntax1: - var1= new Date(year,month,day,hour,minute,seconds,milliseconds);
Example: -
```
const date=new Date(2028,7,28,23,23,34,300);
console.log(date);
console.log(date.getMilliseconds());
```

Syntax2: - var1= new Date(year,month,day,hour,minute,seconds);
Example: -
```
const date=new Date(2028,7,28,23,23,34);
console.log(date);
```

Syntax3: - var1= new Date(year,month,day,hour,minute);
Example: -
```
const date=new Date(2028,7,28,23,23);
console.log(date);
```

Syntax4: - var1= new Date(year,month,day,hour);
Example: -
```
const date=new Date(2028,7,28,23);
console.log(date);
```

Syntax5: - var1= new Date(year,month,day);
Example: -
```
const date=new Date(2028,7,28);
console.log(date);
```

Syntax6: - var1= new Date(year,month);
Example: -
```
const date=new Date(2028,7);
```

console.log(date);

Syntax7: - var1= new Date(year(2 digits),month,day);
//ommiting two digit will get previous century year (19XX)
Example: -
const date=new Date(99,7,20);
console.log(date);

method4: - it take single number argument as milliseconds, it counts milliseconds from 1 january 1970.
Syntax1: - var1= new Date(milliseconds);
Example1: -
const date=new Date(1);
console.log(date);

Example2: -
const date=new Date(1739341884577);
console.log(date);

Date string method: -

toString: - it convert date object into string
Syntax: - date1.toString();
Example: -
const date=new Date();
let str=date.toString();
console.log(str);

toDateString: - it convert date object into string with more readable format.
Syntax: - date1.toDateString();
Example: -
const date=new Date();
let str=date.toDateString();
console.log(str);

toUTCString: - it convert date object into string with UTC (universal time coordinated) standard.
Syntax: - date1.toUTCString();
Example: -
const date=new Date();
let str=date.toUTCString();
console.log(str);

toISOString: - it convert date object into string with ISO (international organisation for standardisation) standard.
Syntax: - date1.toISOString();
Example: -
const date=new Date();
let str=date.toISOString();

console.log(str);

toTimeString: - this method returns the time portion of a date object as a string.
Syntax: - date1.toTimeString();
Example: -
const date=new Date();
let str=date.toTimeString();
console.log(str);

Date format: - date formats for initialise date object.

ISO complete date: - it set as YYYY-MM-DD, it is ISO 8601 is the standard for representation of dates and time.
Syntax: - var1=new Date("YYYY-MM-DD");
Example: -
const date= new Date("2015-03-25");
console.log(date);

ISO year and month date: - it set as YYYY-MM without specifiying day.
Syntax: - var1=new Date("YYYY-MM");
Example: -
const date= new Date("2015-03");
console.log(date);

ISO year only date: - it set as YYYY without specifiying month and day.
Syntax: - var1=new Date("YYYY");
Example: -
const date= new Date("2015");
console.log(date);

ISO date and time(UTC): - it set as YYYY-MM-DDTHH:MM:SSZ (year-month-date-T-hour-minutes-seconds-Z). in format T used for separate Time and Z used for define UTC time.
Syntax: - var1=new Date("YYYY-MM-DDTHH:MM:SSZ");
Example: -
const date= new Date("2021-06-25T12:00:23Z");
console.log(date);
Note1: - UTC (Universal Time Coordinated) is the same as GMT (Greenwich Mean Time).
Note2: - Omitting T or Z in a date-time string can give different results in different browsers.

ISO date and time(UTC relative): - it set as YYYY-MM-DDTHH:MM:SS(+/-)HH:MM (year-month-date-T-hour-minutes-seconds-(+/-)-HH:MM). in format T used for separate Time and use (+/-)HH:MM for define time relative to UTC.
Syntax: - var1=new Date("YYYY-MM-DDTHH:MM:SS(+/-)HH:MM");
Example: -
const date= new Date("2021-06-25T12:00:23+05:30");
console.log(date);

Timezone: - When setting a date, without specifying the time zone, JavaScript will use the browser's time zone.
When getting a date, without specifying the time zone, the result is converted to the browser's time zone.
In other words: If a date/time is created in GMT (Greenwich Mean Time), the date/time will be converted to CDT (Central US Daylight Time) if a user browses from central US.

Short date: - it write as MM/DD/YYYY
Syntax: - var1=new Date("MM/DD/YYYY");
Example: -
```
const date= new Date("05/23/2024");
console.log(date);
```

warning: -

1. In some browsers, months or days with no leading zeroes may produce an error
   Ex: - new Date("5/3/2024");

2. The behavior of "YYYY/MM/DD" is undefined. Some browsers will try to guess the format. Some will return NaN or invalid date.
   Ex: - new Date("2024/05/20");

3. The behavior of "DD-MM-YYYY" is undefined. Some browsers will try to guess the format. Some will return NaN or invalid date.
   Ex: - new Date("20-05-2024");

Long date: - it write as "MMM DD YYYY" format. Month write as abbreviated and moth and day order can be change.
Syntax1: - var1=new Date("MMM DD YYYY");
Example: -
```
const date= new Date("Aug 28 2001");
console.log(date);
```

Syntax2: - var1=new Date("DD MMM YYYY");
Example: -
```
const date= new Date("28 aug 2001");
console.log(date);
```

Long date full month: - it write as "MMM(full) DD YYYY" format. Month write as full name and moth and day order can be change. It is case insensitive.
Syntax1: - var1=new Date("MMM(full) DD YYYY");
Example: -
```
const date= new Date("August 28 2001");
console.log(date);
```

Syntax2: - var1=new Date("DD MMM(full) YYYY");
Example1: - lowercase month name
```
const date= new Date("28 august 2001");
```

```
console.log(date);
```

Example2: - uppercase month name

```
const date= new Date("28 AUGUST 2001");
console.log(date);
```

Date.parse: - it convert valid string date in milliseconds, with these milliseconds can create new date object. It is static method of date object.
Syntax: - var1=Date.parse("string date");
Example: -

```
let msec= Date.parse("March 24 2022");
const date=new Date(msec);
console.log(date);
```

Date get methods: -

getFullYear: - this method returns the year of a date as four digit number.
Syntax: - var1=date.getFullYear();
Example: -

```
const date=new Date();
let year=date.getFullYear();
console.log("Year:-",year);
```

getMonth: - this method returns month of a date as number in range 0-11, as 0 is january and 11 is december.
Syntax: - var1=date.getMonth();
Example: -

```
const date=new Date();
let month=date.getMonth();
console.log("Month:-",month);
```

getDate: - this method returns day of a date as number in range 1-31.
Syntax: - var1=date.getDate();
Example: -

```
const date=new Date();
let dt=date.getDate();
console.log("Date:-",dt);
```

getHours: - this method returns hours of a date as number in range 0-23.
Syntax: - var1=date.getHours();
Example: -

```
const date=new Date();
let hours=date.getHours();
console.log("Hours:-",hours);
```

getMinutes: - this method returns minutes of a date as number in range 0-59.
Syntax: - var1=date.getMinutes();

Example: -
```
const date=new Date();
let mins=date.getMinutes();
console.log("Minutes:-",mins);
```

getSeconds: - this method returns seconds of a date as number in range 0-59.
Syntax: - var1=date.getSeconds();
Example: -
```
const date=new Date();
let secs=date.getSeconds();
console.log("Seconds:-",secs);
```

getMilliseconds: - this method returns milliseconds of a date as number in range 0-999.
Syntax: - var1=date.getMilliseconds();
Example: -
```
const date=new Date();
let msecs=date.getMilliseconds();
console.log("Milliseconds:-",msecs);
```

getDay: - this method returns weekday of a date as number in range 0-6, where 0 is Sunday and 6 is Saturday
Syntax: - var1=date.getDay();
Example: -
```
const date=new Date();
let day=date.getDay();
console.log("Day:-",day);
```

getTime: - this method returns numbers of milliseconds since 1 january 1970.
Syntax: - var1=date.getTime();
Example: -
```
const date=new Date();
let time=date.getTime();
console.log("Time:-",time);
```

Date.now: - this method returns numbers of milliseconds since 1 january 1970. It is static method of date object.
Syntax: - var1==Date.now();
Example: -
```
let now=Date.now();
console.log("Now:-",now);
```

getUTCFullYear: - this method returns the UTC year of a date as four digit number.
Syntax: - var1=date.getUTCFullYear();
Example: -
```
const date=new Date();
let yearUTC=date.getUTCFullYear();
console.log("UTC Year:-",yearUTC);
```

getUTCMonth: - this method returns UTC month of a date as number in range 0-11, as 0 is january and 11 is december.
Syntax: - var1=date.getUTCMonth();
Example: -
```
const date=new Date();
let monthUTC=date.getUTCMonth();
console.log("UTC Month:-",monthUTC);
```

getUTCDate: - this method returns UTC day of a date as number in range 1-31.
Syntax: - var1=date.getUTCDate();
Example: -
```
const date=new Date();
let dtUTC=date.getUTCDate();
console.log("UTC Date:-",dtUTC);
```

getUTCHours: - this method returns UTC hours of a date as number in range 0-23.
Syntax: - var1=date.getUTCHours();
Example: -
```
const date=new Date();
let hoursUTC=date.getUTCHours();
console.log("UTC Hours:-",hoursUTC);
```

getUTCMinutes: - this method returns UTC minutes of a date as number in range 0-59.
Syntax: - var1=date.getUTCMinutes();
Example: -
```
const date=new Date();
let minsUTC=date.getUTCMinutes();
console.log("UTC Minutes:-",minsUTC);
```

getUTCSeconds: - this method returns UTC seconds of a date as number in range 0-59.
Syntax: - var1=date.getUTCSeconds();
Example: -
```
const date=new Date();
let secsUTC=date.getUTCSeconds();
console.log("UTC Seconds:-",secsUTC);
```

getUTCMilliseconds: - this method returns UTC milliseconds of a date as number in range 0-999.
Syntax: - var1=date.getUTCMilliseconds();
Example: -
```
const date=new Date();
let msecsUTC=date.getUTCMilliseconds();
console.log("UTC Milliseconds:-",msecsUTC);
```

getUTCDay: - this method returns UTC weekday of a date as number in range 0-6, where 0 is Sunday and 6 is Saturday
Syntax: - var1=date.getUTCDay();

Example: -
```
const date=new Date();
let dayUTC=date.getUTCDay();
console.log("UTC Day:-",dayUTC);
```

getTimezoneOffset: - this method return difference in minutes between local time and UTC time.
Syntax: - var1=date.getTimezoneOffset();
Example: -
```
const date=new Date();
let diff=date.getTimezoneOffset();
console.log("Difference:-",diff,"mins");
```

Date set methods: -

setFullYear: - this method sets the year of date object. It take year as number in parameter, this method also take optional parameters as month in range 0 to 11 and date in range 1 to 31.
Syntax1: - date1.setFullYear(year);
Example: -
```
const date=new Date();
date.setFullYear(2022);
console.log("set year:- ",date);
```

Syntax2: - date1.setFullYear(year,month,day);
Example: -
```
const date=new Date();
date.setFullYear(2022,1,12);
console.log("set year:- ",date);
```

setMonth: - this method sets the month of date object. It take month parameter as number in range 0 to 11.
Syntax: - date1.setMonth(month);
Example: -
```
const date=new Date();
date.setMonth(0);
console.log("set month:- ",date);
```

setDate: - this method sets the date of date object. It take date parameter as number in range 1 to 31. This method also add days to date object.
Syntax: - date1.setDate(date);
Example1: -
```
const date=new Date();
date.setDate(15);
console.log("set date:- ",date);
```

example2: - add days to date object.
```
const date=new Date();
date.setDate(date.getDate()+5);
```

```
console.log("set date add days:- ",date);
```

setHours: - this method sets the hours of date object. It take hours parameter as number in range 0 to 23.
Syntax: - date1.setHours(hours);
Example: -

```
const date=new Date();
date.setHours(15);
console.log("set hours:- ",date);
```

setMinutes: - this method sets the minutes of date object. It take minutes parameter as number in range 0 to 59.
Syntax: - date1.setMinutes(minutes);
Example: -

```
const date=new Date();
date.setMinutes(15);
console.log("set Minutes:- ",date);
```

setSeconds: - this method sets the seconds of date object. It take seconds parameter as number in range 0 to 59.
Syntax: - date1.setSeconds(seconds);
Example: -

```
const date=new Date();
date.setSeconds(15);
console.log("set Seconds:- ",date);
```

setMilliseconds: - this method sets the milliseconds of date object. It take milliseconds parameter as number in range 0 to 999.
Syntax: - date1.setMilliseconds(milliseconds);
Example: -

```
const date=new Date();
date.setMilliseconds(300);
console.log("set Milliseconds:- ",date);
console.log("Milliseconds:- ",date.getMilliseconds());
```

setUTCFullYear: - this method sets the UTC year of date object. It take year as number in parameter, this method also take optional parameters as month in range 0 to 11 and date in range 1 to 31.
Syntax1: - date1.setUTCFullYear(year);
Example: -

```
const date=new Date();
date.setUTCFullYear(2022);
console.log("set UTC year:- ",date);
```

Syntax2: - date1.setUTCFullYear(year,month,day);
Example: -

```
const date=new Date();
```

```
date.setUTCFullYear(2022,1,12);
console.log("set UTC year:- ",date);
```

setMonth: - this method sets the UTC month of date object. It take month parameter as number in range 0 to 11.
Syntax: - date1.setUTCMonth(month);
Example: -

```
const date=new Date();
date.setUTCMonth(0);
console.log("set UTC month:- ",date);
```

setUTCDate: - this method sets the UTC date of date object. It take date parameter as number in range 1 to 31. This method also add days to date object.
Syntax: - date1.setUTCDate(date);
Example1: -

```
const date=new Date();
date.setUTCDate(15);
console.log("set UTC date:- ",date);
```

example2: - add days to date object.

```
const date=new Date();
date.setUTCDate(date.getUTCDate()+5);
console.log("set UTC date add days:- ",date);
```

setHours: - this method sets the UTC hours of date object. It take hours parameter as number in range 0 to 23.
Syntax: - date1.setUTCHours(hours);
Example: -

```
const date=new Date();
date.setUTCHours(15);
console.log("set UTC hours:- ",date);
```

setUTCMinutes: - this method sets the UTC minutes of date object. It take minutes parameter as number in range 0 to 59.
Syntax: - date1.setUTCMinutes(minutes);
Example: -

```
const date=new Date();
date.setUTCMinutes(15);
console.log("set UTC Minutes:- ",date);
```

setUTCSeconds: - this method sets the UTC seconds of date object. It take seconds parameter as number in range 0 to 59.
Syntax: - date1.setUTCSeconds(seconds);
Example: -

```
const date=new Date();
date.setUTCSeconds(15);
console.log("set UTC Seconds:- ",date);
```

setUTCMilliseconds: - this method sets the UTC milliseconds of date object. It take milliseconds parameter as number in range 0 to 999.
Syntax: - date1.setUTCMilliseconds(milliseconds);
Example: -

```
const date=new Date();
date.setUTCMilliseconds(300);
console.log("set UTC Milliseconds:- ",date);
console.log("UTC Milliseconds:- ",date.getUTCMilliseconds());
```

Date.UTC: - this method get number between 1 jaunary 1970 to given date paramter and return milliseconds according to UTC. it take multiple arguments in numbers for year, month, day, hour, minute, seconds and milliseconds. Adding greater values than given range will overflow in previous value.
Values range: -
Year: - negative or positive values.Ex: - -100, 2028
Month: - 0 to 11 respectively january to december. Ex: - 2(March), 4(May)
Day: - 1 to 31 according to months. Ex: - 3, 7
Hour: - 0 to 23. Ex: - 5, 6
Minute: - 0 to 59. Ex: - 45,57
Seconds: - 0 to 59. Ex: - 30,40
Milliseonds: - 0 to 999. Ex: - 300, 700

Syntax1: - var1= Date.UTC(year,month,day,hour,minute,seconds,milliseconds);
Example: -

```
let ms=Date.UTC(2028,7,28,23,23,34,300);
const date=new Date(ms);
console.log(date);
console.log(date.getMilliseconds());
```

Syntax2: - var1= Date.UTC(year,month,day,hour,minute,seconds);
Example: -

```
let ms=Date.UTC(2028,7,28,23,23,34);
const date=new Date(ms);
console.log(date);
```

Syntax3: - var1= Date.UTC(year,month,day,hour,minute);
Example: -

```
let ms=Date.UTC(2028,7,28,23,23);
const date=new Date(ms);
console.log(date);
```

Syntax4: - var1= Date.UTC(year,month,day,hour);
Example: -

```
let ms=Date.UTC(2028,7,28,23);
const date=new Date(ms);
console.log(date);
```

Syntax5: - var1= Date.UTC(year,month,day);
Example: -

```
let ms=Date.UTC(2028,7,28);
const date=new Date(ms);
console.log(date);
```

Syntax6: - var1= Date.UTC(year,month);
Example: -

```
let ms=Date.UTC(2028,7);
const date=new Date(ms);
console.log(date);
```

Syntax7: - var1= Date.UTC(year);
Example: -

```
let ms=Date.UTC(2028);
const date=new Date(ms);
console.log(date);
```

Compare dates: - compare dates using comparison operators.
Syntax: - var1=date2>date1;
Example: -

```
let text = "";
const today = new Date();
const someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
    text = "Today is before January 14, 2100.";
} else {
    text = "Today is after January 14, 2100.";
}
console.log(text);
```

Destructuring object: - it unpack object properties into variables

Syntax: - let {property1, …, propertyN} = obj1;
Example: -

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 50
};
console.log(person);

let {age, lastName, firstName} = person;
```

```
console.log("Age:-",age);
console.log("Last name:-",lastName);
console.log("First name:-",firstName);
```

Default values: - set values to missing property.
Syntax: - let {property1, missingProperty=value} = obj1;

Example: -

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 50
};
console.log(person);

let {age, lastName, firstName, country="US"} = person;
console.log("Age:-",age);
console.log("Last name:-",lastName);
console.log("First name:-",firstName);
console.log("Country:-",country);
```

Property alias: - it defined variable name for store property value when destructuring object.
Syntax: - let {property1: var1,property2: var2 } = obj1;

Example: -

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 50
};
console.log(person);

let {lastName: lname, firstName: name} = person;
console.log("Last name:-",lname);
console.log("First name:-",name);
```

Destructuring array: - it pick array values into variables.

Syntax: - let [var1, …, varN] = array1;
Example: -

```
const fruits = ["Bananas", "Oranges", "Apples", "Mangos"];
console.log(fruits);

let [fruit1, fruit2] = fruits;
console.log(fruit1);
console.log(fruit2);
```

Skip array values: - use two or more commas for skip values in array.
Syntax: - let [var1,,,var2] = array1;

Example: -
```
const fruits = ["Bananas", "Oranges", "Apples", "Mangos"];
console.log(fruits);

let [fruit1,,,fruit2] = fruits;
console.log(fruit1);
console.log(fruit2);
```

Array position values: - it pick values from specific index from array
Syntax: - let {[index]:var1,[index]:var2} = array1;

Example: -
```
const fruits = ["Bananas", "Oranges", "Apples", "Mangos"];
console.log(fruits);

let {[1]:fruit1,[0]:fruit2} = fruits;
console.log(fruit1);
console.log(fruit2);
```

Spread (…) operator: - it stores remaining array values in another array.
Syntax: - let [var1, var2, ...new_array] = array1

Example: -
```
const numbers = [10, 20, 30, 40, 50, 60, 70];
console.log(numbers);

const [a,b, ...arr2] = numbers
console.log(a);
console.log(b);
console.log(arr2);
```

Destructuring other: -

Map destructuring: - use key value array in map
Syntax: - for (const [key, value] of map1){…}

Example: -
```
const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
]);
```

```
let text = "";

for (const [key, value] of fruits) {
    text += key + " is " + value +",";
}
console.log(text);
```

String destructuring: - destructure string as array
Syntax: - let [var1, …, varN] = str1;

Example: -

```
let name = "good";
let [a1, a2, a3, a4] = name;
console.log(a1);
console.log(a2);
console.log(a3);
console.log(a4);
```

Swap variables: - swap two variables value using a destructuring assignment.
Syntax: - [var1,var2]=[var2,var1];

Example: -

```
let n1=34;
let n2=56;
console.log("before swap n1:-",n1,"n2:-",n2);

[n1,n2]=[n2,n1];
console.log(" after swap n1:-",n1,"n2:-",n2);
```

DEBUGGING: -

Code Debugging: - Programming code might contain syntax errors, or logical errors. Many of these errors are difficult to diagnose.

Often, when programming code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.
Searching for errors in programming and fixing code is called code debugging.

JavaScript Debuggers: - Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.Normally (otherwise follow the steps at the bottom of this page), you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

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: -

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Debugging</h1>
      <script>
         a = 5;
         b = 6;
         c = a + b;
         console.log(c);
      </script>
   </body>
</html>
```

Setting Breakpoints: - In the debugger window, you can set breakpoints in the JavaScript code. At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values. After examining values, you can resume the execution of code (typically with a play button).

The debugger keyword: - The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function. This has the same function as setting a breakpoint in the debugger.

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: -

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Debugging</h1>
      <script>
         console.log("Addition of x and y values");
         debugger;
         let x=45;
         debugger;
         console.log("x:-",x);
         debugger;
         console.log("y:-",y);
         debugger;
         z=x+y;
         console.log("x+y:-",z);
      </script>
```

```
    </body>
</html>
```

Debugging is the process of testing, finding, and reducing bugs (errors) in computer programs.
The first known computer bug was a real bug (an insect) stuck in the electronics.

Errors: - When executing JavaScript code, different errors can occur. Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Try and catch: - The try statement allows you to define a block of code to be tested for errors while it is being executed. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

Generally in JavaScript, if after error occurred, it will stop execution of program after error, but if error occurred in try block, code after try-catch block will execute.

JavaScript has built-in error object to catch error, this object is initialise in catch parameter.

Syntax: -

```
try {
    Block of code to try
}
catch(error_object) {
    Block of code to handle errors
}
```

Example1: - error code without try and catch block.

```
console.log("program complete");
console.log("good day "+a); //Error code
console.log("program complete");
```

Example2: - error code with try and catch block.

```
console.log("program complete");
try{
     console.log("good day "+a); //Error code
}
catch(err)
{
    console.log(err);
}
console.log("program complete");
```

Error object: - JavaScript has a built in error object that provides error information when an error occurs. The error object provides two useful properties: name and message.

Error properties: -

Name: - Sets or returns an error name
Message: - Sets or returns an error message (a string)

Syntax: -

```
try{
    Block of code to try
}
catch(error_object)
{
    error_object.name="value"; //set error object name
    error_object.message="message"; //set error object message

    console.log(error_object.name);   //set error object name
    console.log(error_object.message); //set error object message
}
```

Example: -

```
try{
    console.log("good day "+a);
}
catch(err)
{
    err.name="gooderror";
    err.message="error is good here";

    console.log(err.name);
    console.log(err.message);
}
```

Error Classes: - it has multiple error classes, these classes object throws when error occurred in javascript. This can access with name property of error object.

Error classes name: -

1. **Error: -** it creates error object and it is superclass of other error classes.
   Name: - Error
2. **Eval error: -** An error has occurred in the eval() function. It is depricated use SyntaxError instead.
   Name: - EvalError
3. **Range error: -** this error is thrown if you use a number that is outside the range of legal values.
   Name: - RangeError
4. **Reference error: -** this error is thrown if you use (reference) a variable that has not been declared
   Name: - ReferenceError
5. **Syntax error: -** this error is thrown if you try to evaluate code with a syntax error
   Name: - SyntaxError
6. **Type error: -** this error is thrown if an operand or argument is incompatible with the type expected by an operator or function.

Name: - TypeError

7. **URI (Uniform Resource Identifier) Error: -** this error is thrown if you use illegal characters in a URI function
   Name: - URIError

Example: -

Example1: - Range Error

```
let num=1;
try{
    num.toPrecision(500);
}
catch(err){
    console.log(err.name);
    console.log(err.message);
}
```

Example2: - Reference Error

```
let x=5;
try{
    x=y+1;
    console.log(x);
}
catch(err){
    console.log(err.name);
    console.log(err.message);
}
```

Example3: - Syntax Error

```
try{
    eval("alert('hello)");
}
catch(err){
    console.log(err.name);
    console.log(err.message);
}
```

Example4: - Type Error

```
let num=1;
try{
    num.toUpperCase();
    console.log(num);
}
catch(err){
    console.log(err.name);
    console.log(err.message);
}
```

Throw statement: - The throw statement allows you to create a custom error. Technically you can throw an exception (throw an error). The exception can be a JavaScript String, a Number, a Boolean or an Object

Syntax: -

```
try {
    Block of code to try
    throw Exception("Message of error");
}
catch(error_object) {
    Block of code to handle errors
}
```

Example: -

```
try{
    let a=34;
    let b=0;
    //let b="ram";
    if(b==0)
    {
        throw Error("Divide By Zero");
    }
    let c=a/b;
    if(Number.isNaN(c))
    {
        throw Error("Not a Number");
    }

    console.log(`Division of ${a} and ${b} is ${c}`);
}
catch(err){
    console.log(err.name);
    console.log(err.message);
}
```

Finally statement: - The finally statement lets you execute code, after try and catch, regardless of the result. It is effective in function that return values, generally when function return value it stop executing function code in finally block code execute after return value from function.

Syntax: -

```
try {
    Block of code to try
}
catch(error_object) {
    Block of code to handle errors
```

```
}
finally {
    Block of code to be executed regardless of the try / catch result
}
```

Example: -

```
function division(a,b){
    let f;
    try{
        if(b==0){
            throw Error("Divide By Zero");
        }
        let c=a/b;
        f=true;
        return c;
    }
    catch(err){
        f=false;
        return err;
    }
    finally{
        let res=f?"Program complete":"Error occurred in program";
        console.log(res);
    }
}

let ans;
ans=division(6,3);
console.log(ans);

ans=division(7,0);
console.log(ans);
```

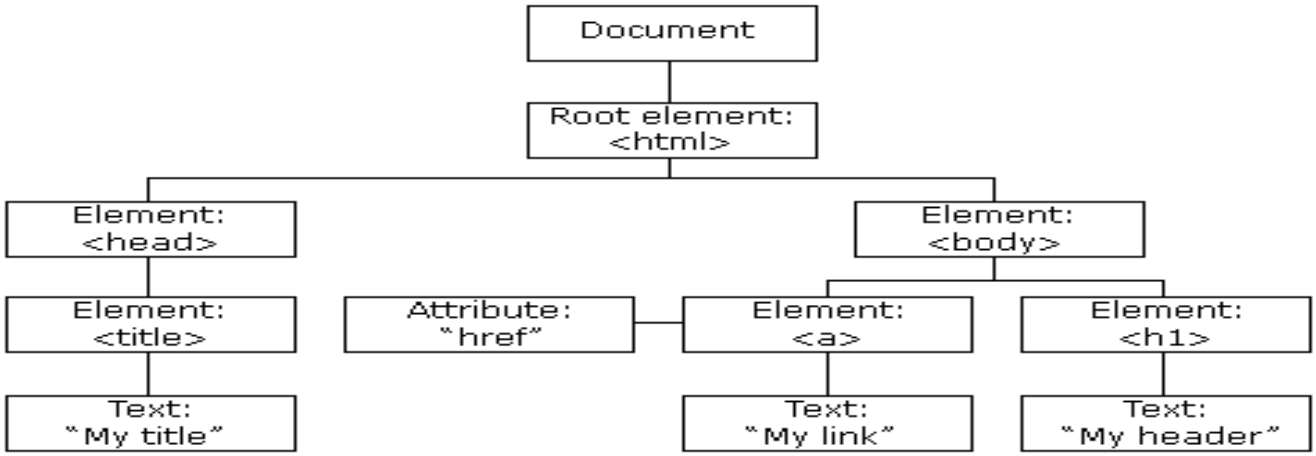
HTML DOM: - document object model

DOM Intro: - When a web page is loaded, the browser creates a Document Object Model of the page. The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

The HTML DOM is a standard for how to get, change, add, or delete HTML elements. With the document object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page



Document Object: - The HTML DOM document object is the owner of all other objects in your web page. The document object represents your web page, for access any element in an HTML page, always start with accessing the document object.

Syntax: - document
Example: -
console.log(document);

DOM html element methods: - these methods use for target html elements. These methods are actions that can perform on HTML Elements.

document.getElementById: - this method find and target element by id, it take string value as parameter. Targeted element return as object in javascript.
Syntax: - var1=document.getElementById("id_name");
Example: -
let head=document.getElementById("head1");
console.log(head);

document.getElementsByTagName: - this method find and target all elements by tag name, it take string value as parameter. Targeted elements return as HTMLCollection of elements.
Syntax: - var1=document.getElementsByTagName("tag_name");
Example: -
let h1Ele=document.getElementsByTagName("h1");

```
console.log(h1Ele);
```

document.getElementsByClassName: - this method find and target all elements by class name, it take string value as parameter. Targeted elements return as HTMLCollection of elements.
Syntax: - var1=document.getElementsByClassName("class_name");
Example: -

```
let h1text=document.getElementsByClassName("text");
console.log(h1text);
```

DOM html element properties: - these properties used for change elements data or get element data.

Element.innerText: - this property used for set inner text in element or get inner text from element.
Syntax1: - var1=Element.innerText;
Example: -

```
let p1=document.getElementById("para1");
let text=p1.innerText;
console.log(text);
```

Syntax2: - Element.innerText="value";
Example: -

```
let p1=document.getElementById("para1");
p1.innerText="Hello world";
```

Element.innerHTML: - this property used for set inner html content in element or get inner html content from element. It parse string values as html elements if valid.
Syntax1: - var1=Element.innerHTML;
Example: -

```
let p1=document.getElementById("para1");
let textHTML=p1.innerHTML;
console.log(textHTML);
```

Syntax2: - Element.innerHTML="HTML value";
Example: -

```
let p1=document.getElementById("para1");
p1.innerHTML="<u>Hello world</u>";
```

Element."**attribute**"**: -** this property used for set or get valid attribute's value of element.
Syntax1: - var1=Element."attribute";
Example: -

```
let lnk1=document.getElementById("lnk1");
let link=lnk1.href;
console.log(link);
```

Syntax2: - Element."attribute"="value";
Example: -

let lnk1=document.getElementById("lnk1");
lnk1.href="2dragdrop.html";

Document properties: - these properties are directly access and return documents elements.

document.baseURI: - this property returns the absolute base URI of the document
Syntax: - var1=document.baseURI;
Example: -
let v=document.baseURI;
console.log(v);

document.body: - this property returns the body element.
Syntax: - var1=document.body;
Example: -
let v=document.body;
console.log(v);

document.doctype: - this property returns document's doctype.
Syntax: - var1=document.doctype;
Example: -
let v=document.doctype;
console.log(v);

document.documentElement: - this property returns html element.
Syntax: - var1=document.documentElement;
Example: -
let v=document.documentElement;
console.log(v);

document.documentURI: - this property returns the URI of the document
Syntax: - var1=document.documentURI;
Example: -
let v=document.documentURI;
console.log(v);

document.embeds: - this property returns the all embed elements from document. embed used for display html page, video, audio or image in html page. With attributes src, type, height and width.
Syntax: - var1=document.embeds;
Example: -
html: -
<embed src="userinput.html" type="text/html" height="35px" width="250px"></embed>

Script: -
let v=document.embeds;

console.log(v);

document.forms: - this property returns the all form elements from document
Syntax: - var1=document.forms;
Example: -
let v=document.forms;
console.log(v);

document.head: - this property returns the head elements
Syntax: - var1=document.head;
Example: -
let v=document.head;
console.log(v);

document.images: - this property returns the all img elements from document
Syntax: - var1=document.images;
Example: -
let v=document.images;
console.log(v);

document.inputEncoding: - this property returns encoding (character set) of document.
Syntax: - var1=document.inputEncoding;
Example: -
Html: - <meta charset="utf-8">
Script: -
let v=document.inputEncoding;
console.log(v);

document.lastModified: - this property returns the date and time when document was updated.
Syntax: - var1=document.lastModified;
Example: -
let v=document.lastModified;
console.log(v);

document.links: - this property returns the all 'a' and 'area' elements from document
Syntax: - var1=document.links;
Example: -
let v=document.links;
console.log(v);

Example2: - img map area example
<img src="https://www.w3schools.com/tags/workplace.jpg" alt="Workplace" usemap="#workmap" width="400" height="379">

<map name="workmap">

```
    <area shape="rect" coords="34,44,270,350" alt="Computer" title="computer"
href="https://www.w3schools.com/tags/computer.htm">
    <area shape="rect" coords="290,172,333,250" alt="Phone" title="phone"
href="https://www.w3schools.com/tags/phone.htm">
    <area shape="circle" coords="337,300,44" alt="Cup of coffee" title="coffee cup"
href="https://www.w3schools.com/tags/coffee.htm">
</map>
```

document.readyState: - this property returns the loading status of document.
Syntax: - var1=document.readyState;
Example: -

```
let v=document.readyState;
console.log(v);
```

Note: - Try this property on console after page loading complete.

document.referrer: - this property returns the URI of the linking document. it will return link of document that clicked to get current page.
Syntax: - var1=document.referrer;
Example: -

```
let v=document.referrer;
console.log(v);
```

Note: - effective result with server.

document.scripts: - this property returns the scripts elements from document
Syntax: - var1=document.scripts;
Example: -

```
let v=document.scripts;
console.log(v);
```

document.title: - this property returns the title of html document
Syntax: - var1=document.title;
Example: -

```
let v=document.title;
console.log(v);
```

document.URL: - this property returns the complete URL of document
Syntax: - var1=document.URL;
Example: -

```
let v=document.URL;
console.log(v);
```

document.all: - this property return all tags from document in html all collection.
Syntax: - var1=document.all;
Example: -

let v=document.all;
console.log(v);

Note: - document.all property is depricated

document.activeElement: - this property return element that currently is in focused.
Syntax: - var1=document.activeElement;
Example: -
let v=document.activeElement;
console.log(v);

document.location: - this property return object with multiple details as hostname, port, pathname etc.
Syntax: - var1=document.location;
Example: -
let v=document.location;
console.log(v);
console.log(v.pathname);

document.writeln: - Writes a string of text followed by a newline character to a document. this method is the same as "document.write()" but adds a newline.
Syntax: - document.writeln("values");
Example1: -
document.writeln("good day");
document.writeln("good day");

Example2: - with "document.write()" method.
document.write("good day");
document.writeln("good day");

Note: - document.write method is depricated

DOM Child and parent : - properties and methods for used for access or manipulate children and parents of elements.

childElementCount: - this property return count of child elements of targeted element.
Syntax: - var1=Element.childElementCount;
Example: -
let v=document.body.childElementCount;
console.log(v);

childNodes: - this property return all child text and element nodes of target element.
Syntax: - var1=Element.childNodes;
Example: -
let v=document.getElementById("box1").childNodes;
console.log(v);

children: - this property return only all child elements of targeted element.
Syntax: - var1=Element.children;
Example: -
```
let v=document.getElementById("box1").children;
console.log(v);
```

firstChild: - this property returns first child node text or element of targeted element.
Syntax: - var1=Element.firstChild;
Example: -
```
let v=document.getElementById("box1").firstChild;
console.log(v);
```

firstElementChild: - this property returns first child element of targeted element.
Syntax: - var1=Element.firstElementChild;
Example: -
```
let v=document.getElementById("box1").firstElementChild;
console.log(v);
```

hasChildNodes: - this method returns true if targeted element have child nodes.
Syntax: - var1=Element.hasChildNodes();
Example: -
```
let v=document.getElementById("box1").hasChildNodes();
console.log(v);
```

lastChild: - this property returns last child node text or element of targeted element.
Syntax: - var1=Element.lastChild;
Example: -
```
let v=document.getElementById("box1").lastChild;
console.log(v);
```

lastElementChild: - this property returns last child element of targeted element.
Syntax: - var1=Element.lastElementChild;
Example: -
```
let v=document.getElementById("box1").lastElementChild;
console.log(v);
```

parentNode: - this property returns parent node or element of targeted element.
Syntax: - var1=Element.parentNode;
Example: -
```
let v=document.getElementById("box1").parentNode;
console.log(v);
```

parentElement: - this property returns parent element of targeted element.
Syntax: - var1=Element.parentElement;
Example: -

```
let v=document.getElementById("box1").parentElement;
console.log(v);
```

DOM Sibling: - properties for access siblings of elements.

nextSibling: - this property return node after targeted element.
Syntax: - var1=Element.nextSibling;
Example: -

```
v=document.getElementById("box1").nextSibling;
console.log(v);
```

nextElementSibling: - this property return element after targeted element.
Syntax: - var1=Element.nextElementSibling;
Example: -

```
v=document.getElementById("box1").nextElementSibling;
console.log(v);
```

previousSibling: - this property return node before targeted element.
Syntax: - var1=Element.previousSibling;
Example: -

```
v=document.getElementById("box1").previousSibling;
console.log(v);
```

previousElementSibling: - this property return element before targeted element.
Syntax: - var1=Element.previousElementSibling;
Example: -

```
v=document.getElementById("box1").previousElementSibling;
console.log(v);
```

DOM Element: -

document.createElement: - this method create HTML element dynamically in javascript. It take element name as string value.
Syntax: - var1=document.createElement("element_name");
Example: -

```
let p1=document.createElement("p");
p1.innerText="Hello world";
console.log(p1);
//console.log([p1]); //see html object proerties
```

append: - this method insert element or text in end of targeted element.
Syntax1: - Element.append(new_element);
Example: -

```
let p1=document.createElement("p");
p1.innerText="Hello world";
```

```
let d1=document.getElementById("d1");
d1.append(p1);
```

Syntax2: - Element.append("string_value");
Example: -

```
let d1=document.getElementById("d1");
d1.append("Good day");
```

appendChild: - this method insert only element in end of targeted element.
Syntax: - Element.appendChild(new_element);
Example: -

```
let p1=document.createElement("p");
p1.innerText="Hello world";
let d1=document.getElementById("d1");
d1.appendChild(p1);
```

prepend: - this method insert element or text in start of targeted element.
Syntax1: - Element.prepend(new_element);
Syntax2: - Element.prepend("string_value");
Example: -

```
let h1=document.createElement("h1");
let d1=document.getElementById("d1");
d1.prepend(h1); //prepent h1 element in div
h1.prepend("Welcome"); //prepent string in h1 element
```

before: - this method insert element or text before of targeted element.
Syntax1: - Element.prepend(new_element);
Syntax2: - Element.prepend("string_value");
Example: -

```
let d1=document.getElementById("d1");
d1.before("good day");
```

after: - this method insert element or text after of targeted element.
Syntax1: - Element.after(new_element);
Syntax2: - Element.after("string_value");
Example: -

```
let d1=document.getElementById("d1");
d1.after("hello world");
```

remove: - this method remove targeted element.
Syntax: - Element.remove();
Example: -

```
let d1=document.getElementById("d1");
d1.remove();
```

removeChild: - this method remove child text or element node of targeted element.

Syntax: - Element.removeChild(child_node);
Example: -
```
let d1=document.getElementById("d1");
d1.removeChild(d1.children[0]);
```

replaceChild: - this method replace child nodes of the targeted elements with new nodes.
Syntax: - Element.removeChild(new_node, old_child_node);
Example: -
```
let h2=document.createElement("h2");
h2.prepend("Hello");
d1.replaceChild(h2,d1.children[0]);
```

DOM Attribute: -

hasAttributes: - this method return true if targeted element has attributes.
Syntax: - Element.hasAttributes();
Example: -
```
let v= document.body.hasAttributes();
console.log(v);
```

Attributes: - this property returns a (NamedNodeMap)collection of attributes in an element.
Syntax: - var1=Element.attributes;
Example: -
```
let img1=document.getElementById("img1");
let img1attr=img1.attributes;
console.log(img1attr);
```

setAttribute: - this property set attribute and value of attribute to targeted element.
Syntax: - Element.setAttribute("name","value");
Example: -
```
let img1=document.getElementById("img1");
img1.setAttribute("src","https://images.pexels.com/photos/674010/pexels-photo-674010.jpeg");
```

getAttribute: - this property return value of attribute to targeted element that passed in parameter.
Syntax: - Element.getAttribute("name");
Example: -
```
let img1=document.getElementById("img1");
let v=img1.getAttribute("width");
console.log(v);
```

createAttribute: - this method creates an attribute node and return as Attr object. Use value property of attribute node (Attr object) to assign value.
Syntax: -
```
var1=document.createAttribute("name");
var1.value="string_value";
```

Example: -
let attr=document.createAttribute("src");
attr.value="https://images.pexels.com/photos/674010/pexels-photo-674010.jpeg";

setAttributeNode: - this method set attribute node to targeted element.
Syntax: - Element.setAttributeNode(attr_node);
Example: -
let img1=document.getElementById("img1");
let attr=document.createAttribute("src");
attr.value="https://images.pexels.com/photos/674010/pexels-photo-674010.jpeg";
img1.setAttributeNode(attr);

getAttributeNode: - this method return attribute node object of passed parameter from targeted element.
Syntax: - var1=Element.getAttributeNode("attribute_name");
Example: -
let img1=document.getElementById("img1");
let v=img1.getAttributeNode("src");
console.log([v]);
console.log(v.value);

removeAttribute: - this method remove attribute that passed to parameter from targeted element.
Syntax: - Element.removeAttribute("attribute_name");
Example: -
let img1=document.getElementById("img1");
img1.removeAttribute("src");

removeAttributeNode: - this method remove attribute node from targeted element that passed as parameter.
Syntax: - Element.removeAttributeNode(attr_node);
Example: -
let img1=document.getElementById("img1");
let v=img1.getAttributeNode("src");
img1.removeAttributeNode(v);

hasAttribute: - this method return true if targeted element has passed attribute as parameter.
Syntax: - Element.hasAttribute("attribute_name");
Example: -
let img1=document.getElementById("img1");
let v= img1.hasAttribute("width");
console.log(v);

DOM Nodes: - in DOM html elements, attributes and texts are nodes.

createTextNode: - this method create text node from string that passed as parameter

Syntax: - var1=document.createTextNode("string_value");
Example: -
```
let v=document.createTextNode("good day");
console.log([v]);

let d1=document.getElementById("d1");
d1.appendChild(v);
```

nodeValue: - this property return node value if have value or return null.
```
let v=document.createTextNode("good day");
console.log([v]);
let value=v.nodeValue;
console.log(value);
```

insertBefore: - this method insert node before referenced node in targeted element.
Syntax: - Element.insertBefore(new_node,reference_node);
Example: -
```
let v=document.createTextNode("Hello world\n");
let cn=d1.childNodes[0];

let d1=document.getElementById("d1");
d1.insertBefore(v,cn);
```

cloneNode: - this method create clone of targeted node. Also can passed true value in parameter for copy same content from targeted node.
Syntax1: - var1=Node.cloneNode();
Example: -
```
let d1=document.getElementById("d1");
let n1=d1.childNodes[1];
let n2=n1.cloneNode();
n2.innerText="Hello world";
d1.append(n2);
```

Syntax2: - var1=Node.cloneNode(true);
Example: -
```
let d1=document.getElementById("d1");
let n1=d1.childNodes[1];
let n2=n1.cloneNode(true);
d1.append(n2);
```

DOM CSS: -

style: - this is element attribute for manipulate css style of element. it access css properties through this attribute. Assign valid string value to that css property.
Syntax: - Element.style.property="value";

Example: -
h1=document.getElementById("title");
h1.style.color="#00f100";

change property with "-" (hyphen) symbol: - as hyphen symbol in indentifiers is not valid in javascript, access css property with hyphen in javascript with captial first letter of word after hyphen symbol in css.
Ex: - (css)background-color, (javascript) style.backgroundColor

Example: -
h1=document.getElementById("title");
h1.style.backgroundColor="black";

querySelector: - this method target element By css selectors, it will return first element that matches with css selector.
Syntax: - var1=document.querySelector("css_selector");
Example: -
let box=document.querySelector(".box");
console.log([box]);

querySelectorAll: - this method target element By css selectors, it will return all element that matches with css selector.
Syntax: - var1=document.querySelectorAll("css_selectors");
Example: -
let boxes=document.querySelectorAll(".box");
console.log(boxes);

use multiple selectors: -
let boxes=document.querySelectorAll("#title, #box1, #box3, #box5");
console.log(boxes);

DOM CLASS: -

className: - this attribute set or get class names of targeted element.
Syntax1: - var1= Element.className;
Example: -
let v=document.querySelector("#title2");
let cn=v.className;
console.log(cn);

Syntax2: - Element.className="class_names";
Example: -
let v=document.querySelector("#title2");
v.className="light font1";

classList: - this property return class names list of targeted element.

Syntax: - var1= Element.classList;
Example: -
```
let v=document.querySelector("#title2");
let cl=v.classList;
console.log(cl);
```

add: - this method add class in class list. It can add multiple classes
Syntax1: - Element.classList.add("class_name");
Example: -
```
let v=document.querySelector("#title2");
v.classList.add("purple");
```

Syntax2: - Element.classList.add("class_name1","class_name2",…,"class_nameN");
Example: -
```
let v=document.querySelector("#title2");
v.classList.add("purple","box","border");
```

remove: - this method remove class in class list. It can remove multiple classes
Syntax1: - Element.classList.remove("class_name");
Example: -
```
let v=document.querySelector("#title2");
v.classList.remove("light");
```

Syntax2: - Element.classList.remove("class_name1","class_name2",…,"class_nameN");
Example: -
```
let v=document.querySelector("#title2");
v.classList.remove("purple","box","border");
```

replace: - this method replace new class name with provided class name in class list.
Syntax: - Element.classList.replace("class_name","new_class_name");
Example: -
```
let v=document.querySelector("#title2");
v.classList.replace("purple","dark");
```

toggle: - this method toggle class in class list.
Syntax: - Element.classList.toggle("class_name");
Example: -
```
let v=document.querySelector("#title2");
v.classList.toggle("purple");
```

Note: - define class at last position in css that will toggle.

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.

Event handling: - define event in html as attribute and assign javascript function call to that event as value.

Event listener: - it is function in JavaScript that waits for an event to accure then respond to it. It is function call that assign to event and execute on event occurrence.

Syntax: -

Html: - <element event_handler="function_name1()" >…</element>
JavaScript: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -

Html: - Click Me
JavaScript: -

```
function good()
{
    alert("good day");
}
```

Event object: - this object contain information about event. This object is initialised in function that invoke with event only. Access object by passing event to function call or use 'event' word in function that will access event.

Syntax1: -
Html: - <element event_handler="function_name1(event)" >…</element>
JavaScript: -

```
function function_name1(event_obj)
{
    //block of code when event invoke
}
```

Example: -
Html: - Click Me
JavaScript: -

```
function good(e)
{
    console.log(e);
}
```

Syntax2: -
Html: - <element event_handler="function_name1()" >…</element>
JavaScript: -

```
function function_name1()
```

```
{
    //block of code when event invoke
    console.log(event);
}
```

Example: -
Html: - Click Me
JavaScript: -

```
function good()
{
    console.log(event);
}
```

Event object properties: - properties for check information.

target: - this property return element that targeted by event object.
Syntax: - var1= event.target;
Example: -
Html: - Click Me
JavaScript: -

```
function good(e)
{
    let v=e.target;
    console.log(v);
}
```

type: - this property return type (name) of event.
Syntax: - var1= event.target;
Example: -
Html: - Click Me
JavaScript: -

```
function good(e)
{
    let v=e.type;
    console.log(v);
}
```

Pass this keyword in event function parameter: - passing this keyword will reference to element that invoke function.

Syntax: -
Html: - <element event_handler="function_name1(this)" >…</element>
JavaScript: -

```
function function_name1(this_elem)
{
    //block of code when event invoke
    console.log(this_elem);
}
```

Example: -
Html: - Click Me
JavaScript: -

```
function good(elem)
{
    console.log(elem);
}
```

Mouse events: - this events occure with mouse interactions

onclick: - this event will occure when left mouse button is clicked.
Syntax: -
Html: - <element onclick="function_name1()" >…</element>
Js: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -
Html: - <h1 onclick="greet()" id="h1">good day</h1>
Js: -

```
function greet()
{
    alert("good day to all");
}
```

ondblclick: - this event will occure when left mouse button is double clicked.
Syntax: -
Html: - <element ondblclick="function_name1()" >…</element>
Js: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -
Html: - <h1 onclick="func1()" id="h1">good day</h1>
Js: -

```
function func1()
{
    alert("welcome here");
}
```

oncontextmenu: - this event will occure when right mouse button is clicked.

Syntax: -
Html: - <element oncontextmenu="function_name1()" >…</element>
Js: -
```
function function_name1()
{
   //block of code when event invoke
}
```

Example: -
Html: - <h1 oncontextmenu="func2()">context menu</h1>
Js: -
```
function func2()
{
   alert("context here");
}
```

onmouseover: - this event will occure when mouse pointer move-over element.
Syntax: -
Html: - <element onmouseover="function_name1()" >…</element>
Js: -
```
function function_name1()
{
   //block of code when event invoke
}
```

Example: -
Html: -
Js: -
```
function func3()
{
   event.target.style.backgroundColor="red";
}
```

onmouseenter: - this event will occure when mouse pointer enter in element.
Syntax: -
Html: - <element onmouseenter="function_name1()" >…</element>
Js: -
```
function function_name1()
{
   //block of code when event invoke
}
```

Example: -
Html: -
Js: -
```
function func3()
{
```

```
    event.target.style.backgroundColor="red";
}
```

onmouseout: - this event will occure when mouse pointer move-out from element.

Syntax: -

Html: - <element onmouseout="function_name1()" >…</element>

Js: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -

Html: -

Js: -

```
function func1(tagname)
{
    alert("leaving "+tagname);
}
```

onmouseleave: - this event will occure when mouse pointer leave element.

Syntax: -

Html: - <element onmouseleave="function_name1()" >…</element>

Js: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -

Html: -

Js: -

```
function func1(tagname)
{
    alert("leaving "+tagname);
}
```

onmousemove: - this event will occure when mouse move on element.

Syntax: -

Html: - <element onmousemove="function_name1()" >…</element>

Js: -

```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -

Html: -
Js: -
```
let a=0;
function func2()
{
    event.target.innerText=a;
    a++;
}
```

onwheel: - this event will occure when mouse wheel scroll on element.
Syntax: -
Html: - <element onwheel="function_name1()" >…</element>
Js: -
```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -
Html: -
Js: -
```
let c=0;
function func3()
{
    event.target.innerText=c++;
}
```

onmousedown: - this event will occure when mouse button get pressed (click-down) on element.
Syntax: -
Html: - <element onmousedown="function_name1()" >…</element>
Js: -
```
function function_name1()
{
    //block of code when event invoke
}
```

Example: -
Html: -
Js: -
```
function good()
{
    event.target.style.backgroundColor="red";
}
```

onmouseup: - this event will occure when mouse button get released (click-up) on element.
Syntax: -

Html: - <element onmouseup="function_name1()" >…</element>
Js: -

```
function function_name1()
{
   //block of code when event invoke
}
```

Example: -
Html: -
Js: -

```
function good()
{
   event.target.style.backgroundColor="white";
}
```

Action on perticular mouse button pressed: - use button property from mouse event to perform action on perticular button click.
Mouse button: -
event.button values: -

- Left button: - 0
- Wheel button: - 1
- Right button: - 2

Syntax: -
Html: - <element onmousedown="function_name1(event)" >…</element>
Js: -

```
function function_name1(e)
{
   //block of code when event invoke
   var1=e.button;
}
```

Example: -
Html: -
Js: -

```
function key1(e)
{
   if(e.button==0)
   {
      console.log("left button pressed");
   }

   if(e.button==1)
   {
      console.log("wheel button pressed");
   }
```

```
    if(e.button==2)
    {
        console.log("right button pressed");
    }
}
```

Special key combination action with mouse button: - in event object there is properties for special keys, to check that key is pressed or not with mouse event ans set action that keys.

Special keys: -

- Alt key: - event.altKey
- Control key: - event.ctrlKey
- Meta key (window/command (mac) key): - event.metaKey
- Shift key: - event.shiftKey

Syntax: -
Html: - <element onmousedown="function_name1(event)" >…</element>
Js: -

```
function function_name1(e)
{
    //block of code when event invoke
    var1=e.specialKey;
}
```

Example: -
Html: -
Js: -

```
function key1(e)
{
    if(e.altKey==true && e.button==0)
    {
        console.log("alt key pressed");
    }

    if(e.ctrlKey==true && e.button==0)
    {
        console.log("control key pressed");
    }

    if(e.metaKey==true && e.button==0)
    {
        console.log("meta key pressed");
    }

    if(e.shiftKey==true && e.button==0)
    {
        console.log("shift key pressed");
```

```
    }
}
```

Assign event in javascript: - assign event in javascript with attribute to targeted element, and assign event listener (function) to attribute.

Syntax: -

```
Element.onevent=function_name;
function function_name()
{
    //block of code when event invoke
}
```

Example: -

Html: - <h1 id="head1">Good day</h1>

Js: -

```
const head1=document.querySelector("#head1");
head1.onclick=changeColor;
function changeColor()
{
    head1.style.color="red";
}
```

onevent attribute: - with this attribute anonymous function can be assigned. Assign one parameter to function for access event object. 'this' keyword in this function will reference to element.

Syntax: -

```
Element.onevent=function (event_obj1)
{
    //block of code when event invoke
    console.log(this);
    console.log(event_obj1);
}
```

Example: -

```
const head1=document.querySelector("#head1");
head1.onclick=function (e)
{
    this.style.color="red";
    console.log(this);
    console.log(e);
}
```

Assign arrow function: - arrow functions can also assign to event attribute, with passing event parameter also, but in arrow function 'this' keyword is reference to window object.

Syntax: -

```
Element.onevent=(event_obj1)=>{
    //block of code when event invoke
    console.log(this);
    console.log(event_obj1);
}
```

Example: -

```
const head1=document.querySelector("#head1");
head1.onclick=(e)=>{
    this.style.color="red";
    console.log(this);
    console.log(e);
}
```

Clipboard events: -

oncopy: - this event will occure when, user copy content from targeted element.

Syntax: -

```
Element.oncopy=function(){
    //block of code when event invoke
}
```

Example: -

```
let box2=document.getElementById("d2");
box2.oncopy= function(e){
    alert("data copied");
}
```

oncut: - this event will occure when, user cut content from targeted element.

Syntax: -

```
Element.oncut=function(){
    //block of code when event invoke
}
```

Example: -

```
let box2=document.getElementById("d2");
box2.oncut= function(e){
    alert("data cut");
}
```

onpaste: - this event will occure when, user paste content from targeted element.

Syntax: -

```
Element.onpaste=function(){
    //block of code when event invoke
}
```

Example: -

```
let text=document.querySelector("#text");
text.onpaste=function(e){
    alert("data pasted");
}
```

Event methods: -

addEventListener: - this method add event listener (function) for perticular event to targeted element. in this method first pass event name as string without 'on' word, and then passed function to second element, it can take true or false for use capture as optional argument. Default value of use capture is false.

Syntax: -

```
Element..addEventListener("event_name",function_name1,useCapture(optional));
function function_name1(event_obj)
{
    //block of code
}
```

Example1: -

```
btn1.addEventListener("click",click1);
function click1(e)
{
    console.log("button is click");
}
```

Example2: - assign anonymous function to addEventListener method

```
btn1.addEventListener("click",function(e)
{
    console.log("button is click");
});
```

Note: - it can also assign arrow function

Event bubbling: - it is event listener (function) invoking when child element event occure and it invoke event listener (function) of parent and ancestor elements of child element for same event if it assign.

Example: -

Html: -

```
<div id="d1">
    div
    <p id="p1">
        paragraph
        <button id="b1">click here</button>
```

```
    </p>
</div>
```
Js: -
```
let div1=document.getElementById("d1");
let btn1=document.getElementById("b1");
let para1=document.getElementById("p1");

let value=1;

btn1.addEventListener("click", function (e){
console.log("button",value);
value++;
});

para1.addEventListener("click", function(e){
    console.log("paragraph",value);
    value++;
});

div1.addEventListener("click", function(e){
    console.log("div",value);
    value++;
});
```

Output: -
button 1
paragraph 2
div 3

Event capturing: - it is opposite concept of event bubbling. When targeted element's event occure, it starts invoking of event listener (function) for same event as targeted element from first ancestor (document) to targeted element. Enable event capturing using addEventListener method with true value to useCapture parameter

Syntax: -
```
Element..addEventListener("event_name",function_name, true (useCapture));
```

Example: -
Html: -
```
<div id="d1">
    div
    <p id="p1">
        paragraph
        <button id="b1">click here</button>
    </p>
</div>
```
Js: -

```
let div1=document.getElementById("d1");
let btn1=document.getElementById("b1");
let para1=document.getElementById("p1");

let value=1;

btn1.addEventListener("click",click1,true);
function click1(e)
{
    console.log("button",value);
    value++;
}

para1.addEventListener("click",click2,true);
function click2(e)
{
    console.log("paragraph",value);
    value++;
}

div1.addEventListener("click",click3,true);
function click3(e)
{
    console.log("div",value);
    value++;
}
```

Output: -
div 1
paragraph 2
button 3

stopPropagation: - it prevent propagation of same event from being occuring. This method can call with event object.
Propagation: - it means bubbling up to parent elements or capturing down to child elements of event.

Syntax: -

```
Element..addEventListener("event_name",function_name1,useCapture(optional));
function function_name1(event_obj)
{
    //block of code
    event_obj.stopPropagation();
}
```

Example: -
Html: -

```
<div id="box1">
    box1
    <span id="slogan">good day to all</span>
</div>
```
Js: -
```
let box1=document.getElementById("box1");
let slg1=document.getElementById("slogan");

slg1.addEventListener("click",func1);
function func1(e)
{
    console.log("slogan");
    e.stopPropagation();
}

box1.addEventListener("click",func2);
function func2(e)
{
    console.log("box");
    e.stopPropagation();
}
```

stopImmediatePropagation: - this method prevent other event listeners (functions) of the same event being invoke. This method can access with event object.

Syntax: -
```
Element..addEventListener("event_name",function_name1,useCapture(optional));
function function_name1(event_obj)
{
    //block of code
    event_obj.stopImmediatePropagation();
}
```

Example: -
```
let box1=document.getElementById("box1");
box1.addEventListener("click",func1);
function func1(e)
{
    console.log("box");
    e.stopImmediatePropagation();
}

box1.addEventListener("click",func11);
function func11(e)
{
    console.log("box again");
    e.stopImmediatePropagation();
}
```

preventDefault: - this method prevent default behaviour of event when it occure. This method can access with event object.

Syntax: -

```
Element..addEventListener("event_name",function_name1,useCapture(optional));
function function_name1(event_obj)
{
    //block of code
    event_obj.preventDefault();
}
```

Example: - prevent open context menu when click on element.

```
let box1=document.getElementById("box1");
box1.addEventListener("contextmenu",function(e){
    e.preventDefault();
    console.log("contextmenu click");
});
```

removeEventListener: - this method remove event listener (function) from event, it take two parameters as event name and event listener (function) name.

Syntax: - Element.removeEventListener("event_name",function_name);

Example: -

```
let box=document.getElementById("box");
box.addEventListener("click",alertbox);

function alertbox(e){
    alert("box clicked");
}

box.addEventListener("contextmenu",function(e){
    e.preventDefault();
    box.removeEventListener("click",alertbox);
    console.log("alertbox method is removed from event listener");
});
```

Keyboard Events: - these event work with keyboard on body element

onkeydown: - this event occure when keyboard key press

Syntax1: -

```
<body onkeydown='function_name()'>
```

Syntax2: -

```
var1=document.body
var1.onkeydown=function(e){
    //block of code
}
```

Syntax3: -

```
var1=document.body
var1.addEventListener("keydown",function(e){
    //block of code
});
```

Example: -

```
let body=document.body;
body.addEventListener("keydown",function(e){
    body.style.backgroundColor="red";
    console.log(e);
});
```

onkeyup: - this event occure when keyboard key release

Syntax1: -

```
<body onkeyup='function_name()'>
```

Syntax2: -

```
var1=document.body
var1.onkeyup=function(e){
    //block of code
}
```

Syntax3: -

```
var1=document.body
var1.addEventListener("keyup",function(e){
    //block of code
});
```

Example: -

```
let body=document.body;
body.addEventListener("keyup",function(e){
    body.style.backgroundColor="white";
    console.log(e);
});
```

Action on perticular key press: - use key and code property of event object for perform action on perticular key press.

Syntax: -

```
var1=event_obj.key;
var2=event_obj.code;
```

Example: -

```
let body=document.body;
body.addEventListener("keydown",function(e){
    if(e.code=='KeyF')
    {
        body.style.backgroundColor="red";
    }
```

```
    if(e.code=='KeyG')
    {
        body.style.backgroundColor="green";
    }
    if(e.code=='KeyH')
    {
        body.style.backgroundColor="blue";
    }
    console.log(e);
});
```

Action on special key combination with key press: - use special key combination for perform perticular action

Special keys: -

- Alt key: - event.altKey
- Control key: - event.ctrlKey
- Meta key (window/command (mac) key): - event.metaKey
- Shift key: - event.shiftKey

Example: -

```
let body=document.body;
body.addEventListener("keydown",function(e){
    if(e.code=='KeyF' && e.shiftKey==true)
    {
        body.style.backgroundColor="maroon";
    }
    if(e.code=='KeyG' && e.ctrlKey==true)
    {
        body.style.backgroundColor="greenyellow";
    }
    if(e.code=='KeyH' && e.altKey==true)
    {
        body.style.backgroundColor="royalblue";
    }
    console.log(e);
});
```