Java Script

~~HTML elements~~

Javascript is designed by Brendan Eich for Netscape in early 1995 and became an ECMA standard in 1997.
It is also called Live script because of its dynamic nature.

It is an scripting language. (convert high level instruction to machine-level-language.)
Javascript is interpreted at runtime by client browser.

Server Side - means that processing takes place on a web server
Ex- Server Side language - PHP, ASP.NET etc
Client Side means that the processing takes place on users computer. It require browsers to run the script on the client machine without involving any processing on the server.
Ex- Client-Side language - HTML, CSS, JS etc

Some Methods of Java Script

1, getElementById() -

(a)- document.getElementById("demo").innerHTML = "Hello JS";

dot(.)

It finds the element with id = demo.

dot(.)

change the element content to Hello JS

It means JS can change HTML content.

(b)- document.getElementById("demo").src = "pic_bulbon.gif";

attribute of an element with id = demo.

It means JS can change the attribute values.

To change the CSS property

(c) document.getElementById("demo").style.fontsize = "35px";

It means JS can change HTML style or CSS property

(d) document.getElementById("demo").style.display = "none";

It means JS can hide or show HTML element depend on their value of display property

Ways to add JS in webpage

(1) - Externally - by using src attribute of <script> tag

<script src="/path to script file.js"> </script>

creating an external js file with the .js extension and then load it within the page using src attribute of <script> tag

(2) - Inline -

Placing the JS code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload etc

<button onclick="document.getElementById("demo").innerHTML = 'Hello JS'" id="demo">
click me </button>

(3) Internally - Embedding the JS code between a pair of <script> and </script> tag

<script>
//js code
</script>

It is called embedding the JS code

You can place the <script> tag either in <head> tag or <body> tag (at last to better understand & improves display speed).

JavaScript Overview -

1, JavaScript uses Unicode character set and so allows almost all characters, punctuations and symbols.
2, JS is a case-sensitive language.
3, JS (statement means instruction or programming instruction).
4, JS statements are seperated by semicolon.
5, JS ignores multiple spaces and tabs.
6, Comments -
   single line comment - // comment
   multiple line comment - /* comment */
7, JS supports both single quotes and double quotes (" ")
8, Keywords - are the reserved words in JS, which can't be used as variable names or function names.
   Some of keywords are -
   var, let, for, else, if, while, function etc.
9, Identifiers are name given to variables.
   There are some rules to naming same as C, C++.
10, Method to join multiple words into one variable name -
   1, using underscore e.g., first_name, last_name, inter_city.
   2, Upper Camel Case e.g., FirstName, LastName, InterCity.
   3, Lower Camel Case e.g. firstName, lastName, interCity.

JS programmers tend to use lower camel case.

Display Popup Message

JS provides different built-in functions to display popup messages for different purposes (e.g. (1) to display a message and take user's confirmation on it or (2) to display a simple message or (3) display a popup to take user's input value.

1, **alert()** - display a message with ok button.

```
alert("This is an alert box"); //display string
alert(100);                    //display number
```

2, **confirm()** - used to take user's confirmation to proceed.
display a popup message with two buttons Ok and ~~Cancel~~ Cancel

```
var userPreference;
if (confirm("Do u want to save changes?")==true)
{
    userPreferance = "Data saved successfully";
} else {
    userPreference = "Save Cancelled";
}
```

3, **prompt()** - used to take user's input.
display a popup message with input box and two buttons - Ok and cancel option)

```
var z;
z = prompt("store a value");
```

JavaScript Output Methods

(1)- document.write() - writing into HTML output
It should only be used for testing.

```
<script>
    document.write(5+6);
</script>
```

Note- Using document.write() after an HTML document is loaded, will delete all existing HTML

Ex-

```
<button onclick="document.write(5+6)">click</button>
```

When u click on the button, it will delete all the output exept it

(2)- console.log()- For debugging purposes, u can call the console.log() method in the browser to display data.

```
<script>
    console.log(5+6); // Not show output on document
</script>
```

/* F12 on ur keyboard will activate debugging. Then, select console in the debugger menu. Then click Run again */

JavaScript Print

JS does not have any print object or print methods.
U can't access output devices from JS.
The only exception is that u can call the window.print() method in the browser to print or save the content of current window.

<button onclick="window.print()"> Print this page
</button>

When u click on the button (Print this page), it will make the page saved or copied.

JS Variable

Variables are the containers, used to store data(values).
There are 3 ways to declare a JS variable-

(i) Using var
(ii) Using let
(iii) Using const

Variable using var

Ex-①

```
<script>
    document
    var x = 5;          // declaration of variable x with initialization
    var y = 6;
    var z = x + y;
    document.write("The value of z is:" + z);
</script>
```

+ used to join the statements

Ex-②

```
<script>
    var car;            // declaration of variable car
    document.write(car); // It will show undefined
</script>
```

Note- If u don't assign a variable, then its value will be undefined technically

Note- Variables declared with the var keyword can not have block scope means it can access from outside of block

Ex③- Redeclaring JS variable
If u re-declare a JS variable, it will not lose its value.

```
<script>
    var car = "indigo";
    var car;
    document.write(car);        // indigo.
</script>
```

variable using let

1, The let keyword was introduced in ES6 (2015).
2, Variables defined with let can't be redeclared.
Ex- let x = "John Doe";
let x = 0; // Syntax Error: 'x' has already been declared.

Note- Can't be redeclare but reassign the new value as it is a variable.
Ex- let x = 10;
x = 0;
document.write(x); // 0

3, Variables declared inside a { } block with the let keyword can't be accessed from outside the block but var can.

Ex-

```
{
  let x=2;
}
// x can not be used here
```

```
{
  var x=2;
}
// x can be used here
```

```
var x = 10;
// Here x is 10;
{
  var x=2;
  // Here x is 2;
}
// Here x is 2;
```

(4) Redeclaring a variable with let inside a block can't redeclare the variable outside the block.

```
Ex-  let x=10;
     // Here x is 10
     {
       let x=2;
       // Here x is 2;
     }
     // Here x is 10;
```

(5) With lets, redeclaring a variable in the same block is not allowed:

```
Ex-  var x = 2;        // Allowed
     let x = 3;        // Not allowed
     {
       let x = 2;      // Allowed
       let x = 3;      // Not allowed
     }
     {
       let x=2;        // Allowed
       var x=3;        // Not allowed
     }
```

Variable with const

1, The const keyword was introduced in ES6 (2015).
2, Variables defined with keyword const can't be redeclared in same block.
3, Variables defined with const can't be reassigned.

Ex-

```
const PI = 3.141592;
PI = 3.14;        // error.
```

4, JS const variables must be assigned a value when they are declared.

```
const PI = 3.14;      // correct

const PI;             // Incorrect
```

5. Const keyword does not define a constant value. It defines a constant reference to a value.

6. Always use const when you declare - a new array, a new object, a new function, a new RegExp.

7. Using const, you can't reassign a constant array, a constant object but change its property or value or array's element.

Ex-

```
Const cars = ["Saab", "Volvo", "BMW"]; // creating array
cars[0] = "Toyota"; // changing of an element.
cars.push("Audi"); // Adding a new element
```

Ex-

```
Const cars = ["Saab", "Volvo"];
cars = ["Volvo", "Saab", "Toyota"]; // Error
                                    can't reassign.
```

8. Variables defined with const have block scope.

JS Operators

An operator - performs some operation on single or multiple operands

1. Arithmetic operators (+, -, *, **, /, %, ++, --) ← Exponentiation (ES6) ← Remainder
2. Assignment operators (=, +=, -=, *=, **=, /=, %=)
3. Comparision operators (==, ===, !=, !==, >, <, >=, <=, ?)
4. Logical operators (&&, ||, !) logical or
5. Bitwise operators (&, |, ~, ^, <<, >>, >>>)
6. Type operators (typeof, instanceof)

* Exponentiation operator - It raises the first operand to the power of second operand.

```
let x = 5;
let z = x ** 2          // z = 25
    z = Math.pow(x, 2)  // z = 25
```

* Exponentiation Assigned Operator (**=)

x **= y ⇒ x = x^y

* Comparision Operator

== Equal to
=== Equal value and equal type

Ex-
var x = 5; c = "5";

x == c; // returns true because both have same value
x === c; // returns false because of different type

Bitwise operators

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	5 << 1	0101 << 1	1010	10
>>	Right shift	5 >> 1	0101 >> 1	0010	2

* Type of operator
var car;
type of car; // returns undefined

var x = 5;
typeof x; // returns number

let y = "Raj";
type of y; // returns string

JS Data types

Data types indicates the type of data whether it is numeric, alphabetic, boolean, undefined etc. JS includes primitive and non-primitive data types as per ~~latest~~ ECMA script 5.1

Primitive data types-

1. String — var x = "Ram";
2. Number — var x = 64;
3. Boolean — var x = true;
4. Null — var x = null;
5. Undefined — var x;

Non-primitive data type

1. Array — const x = ["Saab", "Volvo"];
2. Object — const x = {fn: Raj, ln: Gupta, age: 19};
   (fn, ln, age: name; Raj, Gupta, 19: value)

Note- JS is a dynamic or loosely typed language because a same variable can hold different data types

Ex let x;
x = "hi"; // string
x = 5; // number

JS Function

1, A function is a block of code designed to perform a particular task.
2, JS function are defined with the "function" keyword.
3, We can use a function declaration or a function expression.

Function declaration -

(1) function without parameters -

Syntax - function fun_name() {

//code to be executed

}

(2) function with parameters -

Syntax -

function fun_name (parameters)
{

//code to be executed

}

Note - Declared functions are not executed immediately. Declared function will be executed when they are invoked (called).

Function Expression -

A JS function can also be defined using an expression.
A function expression can be stored in a variable.
After a function expression has been stored in a variable, the variable can be used as a function.

☆ Function stored in variables do not need function names. The function without fun_name is called anonymous function. They are always invoked (called) using the variable name.

anonymous function

```
const x = function (a,b){ return a*b };
document.write (x);          // fun____ a^b};
document.write (x(4,3));     // 12
```

x(4,3) → calling of function

4, Function can also be defined with a built-in JS function constructor called Function().

```
const myfun = new Function ("a", "b", "return a^b");
let z = myfun (4, 3);
document.write (z);          // 12
```

Note- Function () using is complicated in programs. Most of the time, you can avoid the using the 'new' keyword.

5, Self-Invoking Functions-
Function expression can be made "Self-invoking". A self invoking expression is invoked automatically, when followed by ().

```
(function (){
    let x = "Hello";
})();
```

Function are Objects
The 'type of' operator in JS returns "function" for functions. But really, they are objects as it have both properties and methods.
The argument.length property returns the number of arguments recieved when the function was invoked

```
function myF (a,b){ return arguments.length; }
```

Parameters are the name listed in the function definition.

Arguments are the real values passed to (and recieved by) the function.

Parametin Rules

1, Don't specify data types for parameters.
2, We can't check type of passed arguments.
3, JS functions don't check the number of arguments recieved.
4, If a function is called with missing argument, the missing values are set to undefined but it is better to assign a default value.

Ex -
```
function my(x,y){
    if(y==undefined)
    {  y=2;          //assign a default value
    }
    return x+y;
}
z= my(2);          //missing of a argument
```

You can also assign default value at declaration.
```
function my(x,y=2){
    //code to be executed or statements
}
```

Arguments Object

1, JS functions have a built-in object called the 'arguments'.
2, This object contains an array of arguments used when the function was called (invoked).

Ex-① - // To find a max arguments passed to function

```
function findmax () {
    let max = -Infinity;
    for (let i = 0; i < arguments.length; i++)
    { if (arguments[i] > max)
        { max = arguments[i];
        }
    }
    return max;
}
```

arguments[0] arguments[7]

```
findmax (1, 12, 3, 500, 115, 44, 88, 22);
```

Ex-② - // To find sum of arguments

```
function sum () {
    var sum = 0;
    for (i=0; i < arguments.length; i++)
    {
        sum += arguments[i];
    }
    return sum;
}

sum (4, 5, 9, 2, 0, 5, 4);
```

Note - A JS function can also access the global variable

```
let var=10;
function hi {
    return var*var;
}
```

JS object

Object contains many values in terms of properties and methods. Methods are functions and properties can hold values of primitive data type.

Object can be created in two ways -

1. Object literal is a simple way of creating an object using { } brackets.

```
var obj = {fn: "Raj", ln: "Gupta", age: 20}
```

2. Object constructor (using new keyword.

```
var person = new Object();

// Attach properties and methods to person object

person.firstname = "Raj";
person["lastName"] = "Gupta";
person.age = 20.5;
person.getFullName = function () { return this.firstname
                    + " " + this.lastName; };
```

Access properties and Method -

You can access by two types. only properties

```
objectname.propertyname;
Objectname["propertyname"];
```

You can access the function by only dot operator

```
objectname.method;
```

JS objects are mutable

It means they are addressed by reference, not by value.

Let person is an object.

```
const x = person;   // x does not copy the values of person
                    // object instead of they becomes same
                    // become of same memory allocation
                    // so change in one reflect on other. */

const person = {
    fN: "Raj",
    lN: "Gupta"
    }

x.fN = "Suraj";

person.fN;          // Suraj
```

Nested Objects-

We can assign another object as a property of an object.

```
var person = {
    fN : "Raj",
    lN : "Gupta"
    age : 20.5
    address: {
        id: 1
        country: "India"
        }
    };

person.address.country;     // India
```

JS String

JS string is a primitive data types means it stores primitive value. Primitive value is that which have no property and method.

* JS strings are used to store and manipulate text.
* It must be enclosed in single or double quotation mark.
* To find the length of string, use 'length' property.

ex-

```
var str = "Raj"
str.length;        // 3
```

* To use quotation mark inside a string, use backslash escape character.

\'	'	Single quote
\\	\	Backslash
\"	"	double quote

ex-

```
var str = "My name is 'Raj'";
```

Other escape sequence used in JS are-

\b	backspace
\t	tabulator (horizontal)
\v	tabulator (vertical)
\n	new line

* A string can be used as array:

```
var str = "Raj";
str[0];        // R
str[1];        // a
```

String Objects

string can also be defined as object with the keyword 'new'.

```
let jN = new String("John");
typeof jN;        // object
```

Ex- let x = "John";
let y = new String("John");

// (x == y) is true because both have equal values.
// (x === y) is false because x is string and y is object

Note - * objects can't be compared

Ex
let x = new String("John");
let y = new String("John");

// (x == y) is false because both are objects
// (x === y) is false because both are objects

Some string Methods and properties

(i) length — Ex- string_name.length;
(ii) slice(start, end) — not included — Ex- string_name.slice(6,13);
(iii) substr(start, length)
(iv) substring(start, end) — not included
(v) replace(search value, replace value)
(vi) toUpperCase();

```
var str = "Hello World";
var text = str.toUpperCase();
document.write(text, str);
// HELLO WORLD Hello World
```

(vii) toLowerCase()
(viii) concat(string2) — firststring.concat(" ", joining string);
(ix) padStart(times, placing term) — not included

```
var st = "5";
st.padStart(4, '*');
// ***5
```

(x) padEnd(times, placing term) — not included

(xi) charAt(position);

```
var text = "Hello st";
text.charAt(0); // H
```

(xii) charCodeAt (position) ⇒ returns ASCII value of character

```
var text = "Hello World";
text.charCodeAt(0)    // 72
```

Converting a String to an array

(xiii) split( ) :

Ex-

```
var text = "a,b,c,d,e";
const arr = text.split(",");  // split on comma
document.write(arr[1]);  // b
```

Searching in String

(xiv) indexOf (Searching_string);

```
text.indexOf("World");    // 6
```

(xv) lastIndexOf (Searching_string) ⇒ returns the index of last occurrence of a specified text in String.

```
text.lastIndexOf("o");    // 7
```

(xvi) search (Searching_string)

```
text.search("World")  // 6
```

(xvii) startsWith( ) // returns true if a string begins with a specified value otherwise false

```
text.startsWith("Hello");  // true
text.startsWith("World");  // false
```

(xviii) endsWith( )

JS Numbers

Number type represents integer, float, hexadecimal, octal or exponential value. First character in number must be an integer.

Ex-

```
var int = 100;
var float = 100.5;
var hex = 0xff;
var expo = 123e-5    // 0.00123
var octal = 030
```

Note- Numbers are added. Strings are concatenated.

NaN - Not a Number

It is a JS reserved word indicating that a no is not a legal number.

Note- You can also assign a variable with the value NaN.

```
var x = NaN;
```

typeof returns its ~~value~~ type number

Infinity is a number

```
x = 2/0;      //returns Infinity
y = -2/0;     // -Infinity
```

JS displays number as base 10. But you can use toString() - method to output no. from base 2 to 36.

Ex-

```
let my = 32;
my.toString(10);    //32
my.toString(16);    //20
my.toString(8);     //40
my.toString(2);     //100000
```

Number Method

① toString() - returns a no. as a string

Ex-

```
x = 123;
x.toString();
```

② toExponential() - returns a string, with a number rounded and written using exponential notation

Ex-

```
let x = 9.656;
x.toExponential(2);   // 9.66 e+0   → defines the no. of characters behind the decimal point
x.toExponential(4);   // 9.6600 e+0
x.toExponential();    // 9.656
```

③ toFixed() - returns a string with the number writing with specified no. of decimals

```
let x = 9.656;
x.toFixed(0);   // 10
x.toFixed(2);   // 9.66
```

④ toPrecision() - returns a string with a no. written with specified length

```
let x = 9.656;
x.toPrecision()   // returns 9.656
x.toPrecision(2)  // 9.7
```

⑤ ~~toString~~ valueOf() - returns a no. as a no.

```
let x = 123;
x.valueOf();   // 123
```

Converting Variables to Numbers

3 JS methods used to convert -

Number()
parseInt()
parseFloat()

These are not number methods but global JS methods

Ex- Number (true); // 1
Number ("10"); // 10
Number ("10, 33"); // NaN
Number ("John"); // NaN

Ex- parseInt() parses a string and returns a no. Only the first no. is returned. Spaces are allowed.
parseInt ("10"); // 10
parseInt ("10.33"); // 10
parseInt ("10 years"); // 10
parseInt ("years 10"); // NaN

Number Properties

MAX_VALUE
MIN_VALUE
POSITIVE_INFINITY
NEGATIVE_INFINITY
NAN

number properties belongs to JS's number object wrapper called Number.
These properties can only be accessed as Number.MAX_VALUE

These can't be accessed as
x.MAX_VALUE; // undefined

JS Array

Creating an Array-

Syntax-

```
const array_name = [item1, item2, ...];
```

Ex- `const cars = ["BMW", "Audi", "Indigo"];`

You can also create an array as-

```
const cars = [];
cars[0] = "Saab";
cars[1] = "Volvo";
cars[2] = "BMW";
document.getElementById("demo").innerHTML =
cars;
// Saab, Volvo, BMW
```

Accessing an array Element

```
x = cars[0];        // Saab
```

Changing an array Element

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Indego";
```

Accessing the full array - (using array_name);

```
document.write(cars);
```

Concatenating Array

The concat() method creates a new array by concatenating existing arrays - without changing the merging arrays

Ex-

```
const myGirls = ["hi", "bi", "ni"];
const myBoys = ["hu", "bu", "nu"];
const myChildren = myGirls.concat(myBoys);
```

Arrays Properties and Methods

length	cars.length;	//returns no. of elements
sort()	cars.sort();	// Sorts the array
push()	cars.push("Fortuner");	// Adding a new element
Array.isArray()	Array.isArray(cars);	//returns true,
toString()	cars.toString();	convert an array to string with comma separated
join()	cars.join("*");	It joints all array elements into a string with a given separator
pop()	cars.pop();	// remove last element.
shift()	cars.shift();	// It removes first element
unshift()	cars.unshift("Thor");	add a new element at the beginning and returns new length of array
concat()	arr1.concat(arr2,arr3);	It merges 2 or more array into a new one
reverse()	cars.reverse();	//reverse an array

Arrays Sorting

sort() methods sort the array as a string not as a number

Ex- const cars = ["BMW", "Thor", "Audi"];
cars.sort(); // All
document.write(cars); // Audi, BMW, Thor...

② const no = ["25", 125, 225];
no.sort()
document.write(no); // 225, 225, 125

Thus, It can't sort the numeric array

Numeric Array Sorting

① Using compare function

```
my.sort(function(a, b){return a-b});
```

② Using loops.

Insertion Sorting, Bubble Sorting, etc.

Finding max/min in numeric array

```
Math.max.apply(null, arr_name);
Math.min.apply(null, arr_name);
```

Other Method;

```
function my(arr){
    let len = arr.length;
    let max = -Infinity;
    let min = Infinity;
    while(len-- && len>=0)
    {
        if(arr[len] > max)
            max = arr[len];
        else if(arr[len] < min)
            min = arr[len];
    }
    return max, min;
}
```

JS Dates

Date objects are created with the new Date() constructor.

```
const d = new Date();
```

Dates Methods

When a Date object is created, a no. of methods allow you to operate on it.

Displaying Dates -

toUTCString()	d.toUTCString()
toDateString()	d.toDateString()
toISOString()	d.toISOString()

Let's learn about some date formats -

ISO Date	2015-03-25 (with hyphen)
Short Date	03/25/2022
Long Date	Mar 25 2015

Get Date Methods

getFullYear()	d.getFullYear()
getMonth()	d.getMonth() .0-11
getDate()	d.getDate()
getHours()	d.getHours()
getMinutes()	d.getMinutes()
getTime()	d.getTime()
getDay()	d.getDay()

Set Date Methods

setDate()	d.setDate(23);
setFullYear()	d.setFullYear(2020,11,3);
setHours()	d.setHours(22);
setMinutes()	d.setMinutes(30);
setMonth()	d.setMonth(11);
setTime()	d.setTime()
set	returns the milliseconds since Jan 1970

JS Math object

Syntax -

```
Math.property
Math.method(number)
```

```
Math.PI          // returns PI value 3.141321...
Math.SQRT2       // returns square root of 2

Math.pow(x,y)    // x^y
Math.sin(30)     // 1/2
Math.sqrt(64)    // 8
Math.abs(x)      // returns the positive value of x
```

JS conditions

```
if
if else
if else if
switch
```

```
if
if else
if else if
switch
```

JS loops

① for loop

syntax -

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

② while loop

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

③ do/while loop

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

④ for/in loop

The JS for in statement loops through the properties of an object value of an array

(a) for (key in object)

```
{
    //code to be executed
}
```

for (let x in person)

```
{
    text += person[x];
}
```

⑤ for/of Loop

```
for (variable of iterable)
{
  //code to be executed
}
```

Ex-
```
for (let x of cars)
{
  text += x;
}
```

⑥
```
for (variable in array)
{
  //code to be executed
}
```

```
text = "";
for (let x in cars)
{
  text += cars[x];
}
```

JS Errors

try statement lets you test a block of code for errors
The catch statement lets you handle the error
The throw Statement lets you create custom errors
The finally statements lets you execute code after try and catch, regardless of the result

syntax-
```
try {
  //code to be tested
}
catch (err) {
  document.getElementById("demo").innerHTML = err.message
}
```

Note- try and catch must be in pair.

HTML DOM to access by JS

document.getElementById("id.name");

document.getElementByTagName("tg.nam");

document.getElementByClassName("class-nam");

document.querySelectorAll("p.intro"); - selects a p element with class intro.

SOME DOM EVENTS

onclick
onsubmit
onchange
onmouseover
onmousedown

JS Timing Events

setTimeout(function, milliseconds) - executes a function after waiting a specified number of milliseconds

setInterval(function, milliseconds) - repeats the execution of the function continuously