Fundamentals of JavaScript

What is JavaScript (JS)?

- JavaScript is a client-side as well as server side scripting language that can be inserted into HTML pages and is understood by web browsers. JavaScript is also an Object based Programming language.
- JavaScript is a scripting language that allows you to create dynamically updating content, control multimedia, animate images etc.
- JavaScript is a loosely typed scripting language, that means language does not bother with types too much, and does conversions automatically, i.e. you can easily add string to an integer and get the result as a string.
- The JavaScript is best known as the client side programming language to display the dynamic data on the webpage using the concept called DOM (Document Object Model).
- JavaScript interact with html elements (DOM elements) in order to make interactive web user interface. JavaScript can be used in various activities like data validation, display popup messages, handling different events of DOM elements, modifying style of DOM elements etc.
- JavaScript is a lightweight, interpreted, object-oriented programming language and it is a first class functions.

Lightweight defines that the instructions are executed very faster in client side web browser. The program as to implement very faster and easiest way to implement, and it is a small memory footprint.

Interpreted means In the case of JavaScript there is no compiler to compile the program. The instructions are executed directly one after another.

Object oriented means you can write code that can be re-used. JavaScript is an excellent language to write object oriented web applications. It can support OOP because it supports inheritance through prototyping as well as properties and methods.

First class functions means a **function** can be passed as an argument to other **functions**, can be returned by another **function** and can be assigned as a value to a variable.

```
function sayHello() {
  return "Hello, ";
}
function greeting(helloMessage, name) {
  console.log(helloMessage() + name);
}
// Pass `sayHello` as an argument to `greeting` function
greeting(sayHello, "JavaScript!");
```

We are passing our sayHello() function as an argument to the greeting() function, this explains how we are treating the function as a value.
The function that we pass as an argument to another function, called a Callback function**.** *sayHello is a Callback function.*

Why JavaScript is a lightweight?

JavaScript engines have become a lot more complex, but executing JavaScript code has become a faster.

What is loosely typed scripting?

JavaScript is a loosely typed scripting language, that means language does not bother with types too much, and does conversions automatically, i.e. you can easily add string to an integer and get the result as a string.

Ex:

```
var a = 10;
a++;
console.log(a); // outputs 11
a = 'hello';
console.log(a); // outputs hello
```

A strongly typed language would throw an error at line 4 when you try to re-assign the variable 'a' from an integer to a string, but in loosly typed scripting conversion automatically done.

Ex:

```
//JavaScript Example (loose typing)
var a = 13; // Number declaration
var b = "thirteen"; // String declaration


// Java Example (strong typing)
int a = 13;             // int declaration
String b = "thirteen"; // String declaration
```

What is Non-Browser Environment?

- Initially JavaScript was developed for web browsers only. Later they develop language with many uses and platforms.
- A platform may be a browser, or a web-server, or a washing machine, or another *host*. Each of them provides specific functionality and runs the JavaScript Code on any host.

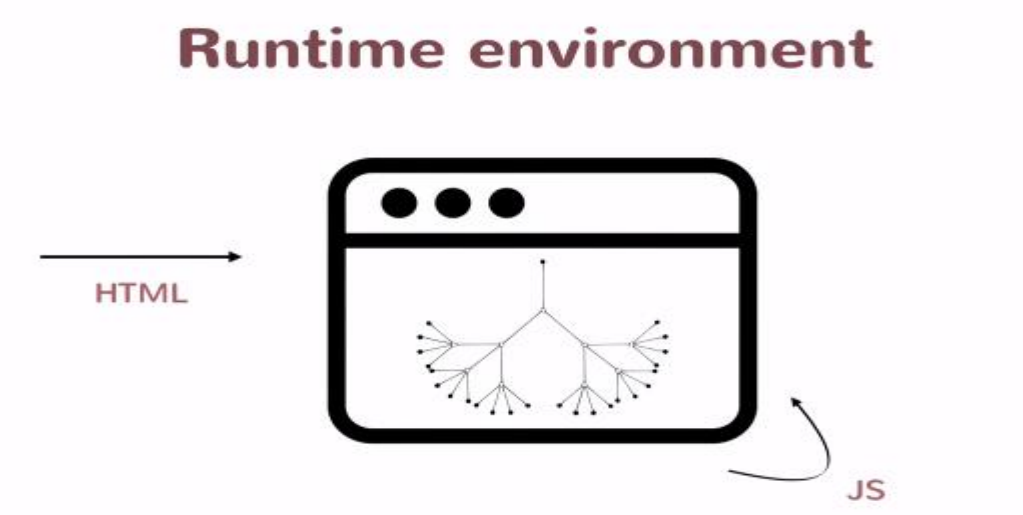
What is Scripting Language?

- JavaScript is a Scripting language, that means basically it's a language write the instructions for the runtime environment.
- Scripting language does not require the compilation step, directly it will executes the instructions at runtime. Instructions are executed one after another.

What is Runtime environment?

- Most popular runtime environment is the web browser. JavaScript runs on a web browser. In the web browser HTML, CSS and JavaScript then the browser renders the HTML and runs the JavaScript. Many web applications Complex pieces of functionalities runs on the client browser.
- We have a web browser making a call to specific web site, we are typing URL in the web browser, then the browser makes the request to the server, then the server responds with HTML content. HTML as we get from the server as a bunch of text or a string of HTML Content.
- Browser looks that text and examines those tags there is an <html> tag, <body> tag, tag, <p> tag etc.takes all those tags and convert them into an objects. For example tag inside

the <p> tag, then it will create the object for tag and create the child object of <p> tag under the object. Like this construct will be happens and making into a tree as called as DOM (Document Object Model) tree.



- Browser interprets the HTML, and builds the DOM tree. Basically HTML is a static if you run n no. of times same DOM tree was getting no changes will be happened on Nodes of DOM tree. However HTML Comes with a JavaScript, Here the JavaScript is a dynamically changes the nodes deleting the nodes from the DOM tree and adding new nodes to the DOM tree and making dynamically. Here the JavaScript most popular.

History of JavaScript

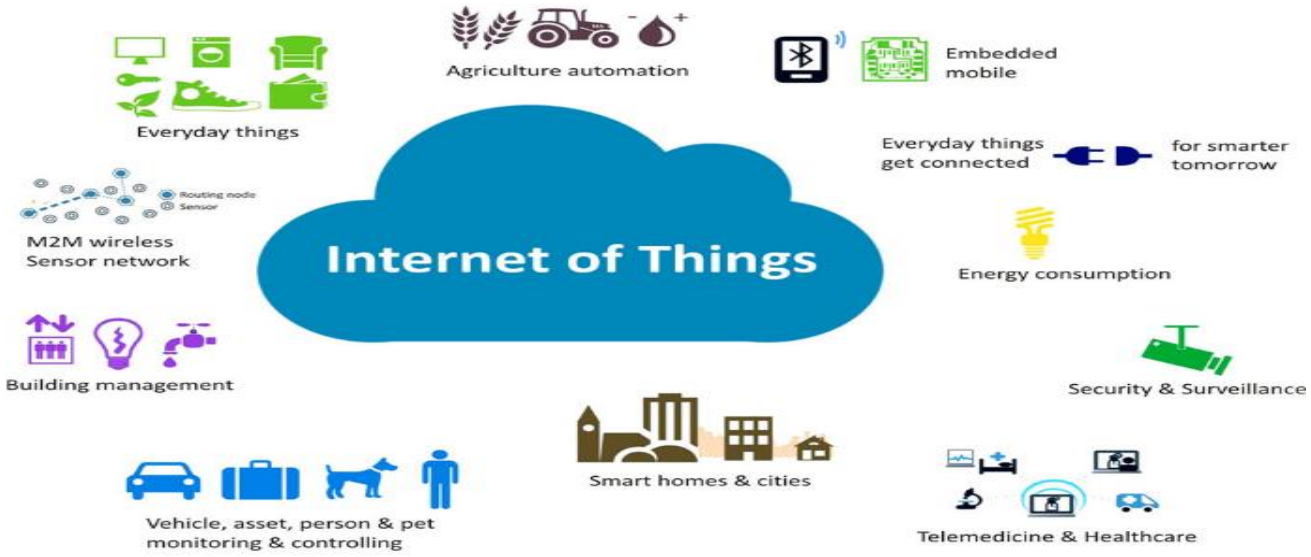
- The JavaScript is developed by Brendan Eich at Netscape Corporation in the year of 1995.
- The first Name of JavaScript is Live Script and later renamed as JavaScript.
- The JavaScript language is standardized by the company called ECMA (European Computers Manufacturing Association). This company is responsible for releasing the versions of ECMA Script.
- The JavaScript is also known as ECMA Script.
- The Current Version of ECMA Script is ES8 or ES-2017. But many projects build on ECMA Script5 why it is happens, till now most of the browsers will not support the latest version. ECMA Script 5 only supported for all the browsers at present.
- We can execute the JavaScript program on any browser without need of HTML.

Features of JavaScript

- Using JavaScript we can develop Client Side Web applications. We know JavaScript we can also use JQuery, Angular JS (Google),React JS (FB).
- Using JavaScript we can develop the server side logic also using Node JS and EXPRESS JS.
- Using JavaScript we can develop the browser extensions *ex:* add-blocker, eyedropper.
- Using JavaScript We can also develop the desktop applications. *ex:* u torrent
- We can develop even Mobile app's by using JavaScript.
- We can also develop the IOT (internet of things) Applications using JavaScript.

What is IoT (Internet of Things)?

- Internet of Things (IoT) is an ecosystem of connected physical objects that are accessible through the internet.
- Objects that have been assigned an IP address and have the ability to collect and transfer data over a network without manual assistance or intervention.
- The embedded technology in the objects helps them to interact with internal states or the external environment, which in turn affects the decisions taken.



Advantages of JavaScript

JavaScript is a client side language: The JavaScript code will be executed on the client's browser in very faster way. It saves bandwidth and load on the web server.

JavaScript language easy to learn: The JavaScript language is easy to learn and implements. It uses the DOM model that provides plenty of predefined functionalities to the various objects on pages.

No compilation needed: JavaScript does not require compilation process so no compiler is needed. The browser interprets JavaScript as it HTML tags.

Platform Independent: If you write JavaScript code then the browser will understand and interprets the JavaScript code. Any JavaScript code can be executed on different types of hardware's.

Disadvantages of JavaScript

- ***Client-Side Security****:* Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable JavaScript.
- ***Browser Support:*** JavaScript is sometimes interpreted differently by different browsers. Whereas server-side scripts will always produce the same output, client-side scripts can be a little unpredictable.
- Don't be overly concerned by this though - as long as you test your script in all the major browsers you should be safe. Also, there are services out there that will allow you to test your code automatically on check in of an update to make sure all browsers support your code.

JavaScript Variables

- JavaScript Variables are used to store the values in the variables.

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

```
var value = 42;
```

- Here, **var** is the *keyword*,
  value is the *variable name* which is used to hold the value
  42 is the *value*
  = operator is the *assignment operator*.
  It means the 42 value will be stored in the browser cache and it is referred with the variable value. Here 42 are assigned to the variable name value.

- Here, don't have the information about the type of the variable whether it is an integer, string or Boolean. So, there is no concept of typed variables in JavaScript. There is no pre-declaration of type required inorder to create variable. We just create a variable and we can assign whatever value you want either it may be a number or string or Boolean.
- For any constant values we have to define with capital letter only.

  Ex: var APP="JS APP";
  var APP_NAME="JS APP";

There are some primitives or primitive Data types to assign the values to variable names.

- Number
- String
- Boolean
- Undefined
- Null
- *Symbol (EMCA Script6)*

Number:

- Number is a JavaScript type for storing numbers.
- Numbers in JavaScript are "double-precision 64-bit format IEEE 754 Values".
- No integers in JavaScript, all are floating point double-precision 64-bit number.

```
var a=20;
var b=30;
var sum=a+b;
console.log(sum);
```

String:

- Strings are sequences of characters; all strings are 64-bit characters.
- There is no character data type, everything is a string in JavaScript, even it is a single character that one also be a string.
- Strings are used to represents text.

```
var fname="satish ";
var lname="andra";
var name=fname+lname;
console.log(name);
```

Boolean:

- Boolean is the primitive data type, Boolean have two values either it may be true or false.
- Boolean values are used for conditionals; inside for loop, while loops and they can affect flow of program.

```
var a=true;
console.log(a);
```

Undefined:

- Before going to know about undefined first we need to know what the difference between declaration and definition is.

What is the difference between declaration and definition?

- Declaration is nothing but create a new variable but not assigning any value to the variable called as declaration.

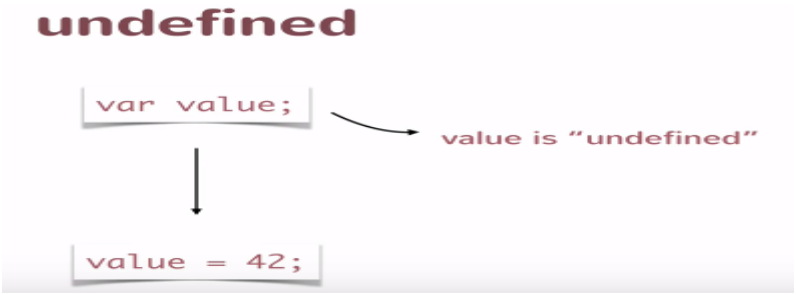
```
var value;
```

- Definition is nothing but create a new variable and assigned a value to the variable name is called as definition.

```
value = 42;
```

Let's know about the undefined

- We are declaring a variable but we are not assigned a value to variable then it is called as undefined.



null:

- In JavaScript null is "nothing". It's just a special value which has the sense of "nothing", "empty" or "value unknown".
- Unfortunately, in JavaScript, the data type of null is an object.

  Ex: var age=null;
- The code above states that the age is unknown or empty for some reason.
- You can consider it a bug in JavaScript that typeof null is an object. It should be null.

JavaScript Undefined vs NULL

- undefined means a variable has been declared but has not yet been assigned a value. On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value.
- Also, undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.
- Unassigned variables are initialized by JavaScript with a default value of undefined. JavaScript never sets a value to null. That must be done programmatically.

```
typeof undefined;    //undefined
typeof null;         //object
null === undefined; //false
null == undefined;   //true
```

Ex

```
// Undefined Example
var abc;
console.log('value ' + abc + ' data type ' + typeof abc);
//value undefined data type undefined

// number example
var jsVersion = 5;
console.log('value ' + jsVersion + ' data type ' + typeof jsVersion);
//value 5 data type number

// String Examples
var empName = 'John';
console.log('value ' + empName + ' data type ' + typeof empName);
//value John data type string

// boolean Examples
var isJSEasy = true;
console.log('value ' + isJSEasy + ' data type ' + typeof isJSEasy);
//value true data type boolean

// null Example
var test = null;
console.log('value ' + test + ' data type ' + typeof test);
//value null data type object

// reassignment example of variables

var someVar;
console.log('value of someVar is : ' + someVar);
//value of someVar is : undefined

someVar = 10;
console.log('value of someVar is : ' + someVar);
//value of someVar is : 10

someVar = "test";
console.log('value of someVar is : ' + someVar);
//value of someVar is : test

someVar = true;
console.log('value of someVar is : ' + someVar);
//value of someVar is : true

someVar = null;
console.log('value of someVar is : ' + someVar);
//value of someVar is : null
```

String Methods

String Methods are used to find the length of the string, and know the index of the string and we will get the substring from the entire string etc…

```
var str = "Please locate where 'locate' occurs!";
```

Method	How to Use	Usage	Result
length	str.length;	length is used to find the length of string.	36
indexOf( )	str.indexOf("locate")	This Method is used to find the index of the given string & it is used to tell the first occurrence of the string.	7
lastIndexOf( )	str.lastIndexOf("locate")	This Method is used to find the last index of the given string.	21
indexOf( )	str.indexOf("locate",15)	This method accept a second parameter as the starting position for the search	21
search( )	str.search("where")	This Method is used to search the string position whether it is present or not. If it is present it returns the current position value, it is not present then it is return -1 value.	14

Ex

```
var str="Please locate where 'locate' occurs!";

//find the length of the string
var findLength=str.length;
console.log("The Length of the String is : "+findLength); //36

//find the index of the string
var indexStr=str.indexOf("locate");
console.log("Index of the String is : "+indexStr);  //7

//find the last index of the string
var lastIndex=str.lastIndexOf("locate");
console.log("Last Index of the String is : "+lastIndex); //21

//find the index but starting position you have to mention
var posIndex=str.indexOf("locate",15);
console.log("find the string where you " +
    "mentioned the position : "+posIndex);  //21

//find the string if it present or not
var searchString=str.search('where');
console.log("String present or not : "+searchString);   //14
```

What is the difference between indexOf() and search() methods.

- The search() method cannot take a second start position argument.
- The indexOf() method cannot take powerful search values (regular expressions).

Extracting String Parts

var str=”Apple, Banana, Kiwi”;

Method	How to Use	Usage	Result

slice(start,end)	str.slice(7,13);
	str.slice(-12,-6);
	str.slice(7);
	str.slice(-12);
		slice() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the start position, and the end position (end not included). This example slices out a portion of a string from position 7 to position 12 (13-1):	Banana
			Banana
			Banana,Kiwi
			Banana,Kiwi

substring(start,end)	str.substring(7,13)	**substring()** is similar to slice(). The difference is that substring() cannot accept negative indexes.	Banana

substr(start,length)	str.substr(7,6);
	str.substr(7);
	str.substr(-4)
		substr() is similar to slice(). The difference is that the second parameter specifies the **length** of the extracted part.	Banana
			Banana,Kiwi
			Kiwi

Ex

```
str="Apple, Banana, Kiwi";
//using slice
console.log(str.slice(7));      //Banana, Kiwi
console.log(str.slice(7,13));   //Banana
console.log(str.slice(-12));    //Banana, Kiwi
console.log(str.slice(-12,-6)); //Banana

//using substring
console.log(str.substring(7));      //Banana, Kiwi
console.log(str.substring(7,15));   //Banana ,

//using substr
console.log("substr.........");
console.log(str.substr(7));     //Banana, Kiwi
console.log(str.substr(7,6));   //Banana
console.log(str.substr(-12));   //Banana, Kiwi
console.log(str.substr(-12,9)); //Banana, K
```

Replacing String Content

- The **replace()** method replaces a specified value with another value in a string.
- By default, the replace() function replaces **only the first** match.

```
str="Please visit Microsoft and Microsoft!";
console.log(str.replace("Microsoft","Tech M"));
//Please visit Tech M and Microsoft!
```

- By default, the replace() function is case sensitive. To replace case insensitive, use a **regular expression** with an **/i** flag (insensitive).

```
str="Please visit Microsoft and Microsoft!";
console.log(str.replace(/MICROSOFT/i,"Google"));
//Please visit Google and Microsoft!
```

- To replace all matches, use a **regular expression** with a **/g** flag (global match).

```
str="Please visit Microsoft and Microsoft!";
console.log(str.replace(/Microsoft/g,"Facebook"));
//Please visit Facebook and Facebook!
```

Converting to Upper and Lower Case

- If you want to convert the string to lowercase or uppercase by lowerCase() & upperCase() functions.
- lowerCase() is used to convert any string to lowercase characters. Whereas upperCase() is used to convert any string to uppercase characters.

```
str="string converting";
console.log(str.toLowerCase()); //string converting
console.log(str.toUpperCase()); //STRING CONVERTING
```

concat() Method

- concat() method is used to join two or more strings into one string.

```
var str1="Hello";
var str2="World";
var concat=str1.concat(" "+str2);
var join=str1+" "+str2;
console.log(concat);  //Hello World
console.log(join);  //Hello World
```

Note: All string methods return a new string. They don't modify the original string. So, Strings are immutable: Strings cannot be changed, only replaced.

String.trim()

- String.trim() removes whitespace from both sides of a string.

```
str = "       Hello World!        ";
alert(str.trim());
```

Extracting String Characters

var str=”Hello World”;

Method	How to Use	Usage	Result
charAt(position)	str.charAt(0);	The charAt() method returns the character at a specified index (position) in a string.	H
charCodeAt(position)	str.charCodeAt(0)	The charCodeAt() method returns the unicode of the character at a specified index in a string. The method returns a UTF-16 code (an integer between 0 and 65535)	72
property Access[ ]	str[0]	This method returns the character at a specified index (position) in a string. ECMAScript 5 (2009) allows property access [ ] on strings.	H

Property access might be a little unpredictable:

- It does not work in Internet Explorer 7 or earlier
- It makes strings look like arrays (but they are not)
- If no character is found, [ ] returns undefined, while charAt() returns an empty string.
- It is read only. str[0] = "A" gives no error (but does not work!)

*If you want to work with a string as an array, you can convert it to an array. A string can be converted to an array by using **split()** method*

Ex

```
//convert String to an array
var str4="Banana,Graps,Apple,Orange";
var arr_str=str4.split(",");
console.log(arr_str[2]);    //Apple
```

Ex

```
/* ---------------------------------------------------------------------
                            String Object Examples
 --------------------------------------------------------------------- */
var msg = "Good Morning";

console.log('Uppercase : ' + msg.toUpperCase());     //GOOD MORNING
console.log('lowercase : ' + msg.toLowerCase());     //good morning

console.log('length : ' + msg.length);          //12

// partial string
var partialStr = msg.substr(0,4); // Good
partialStr = msg.substr(5);
console.log(partialStr);         //Morning

// char @ any index
var ch = msg.charAt(4);
console.log(ch+" Space Here");  //

// reverseString
str = "Good Morning";
function revString(str) {
    var temp = '';
    for(var i = str.length-1; i>=0 ; i--){
        var ch = str.charAt(i);
        temp += ch;
    }
    return temp;
}
console.log('Rev String : ' + revString(str));  //gninroM dooG
// Palindrome
str = "ABCDCBA";
function palindrome(str) {
    return str === revString(str);
}

console.log('is Palindrome ? ' + palindrome(str));
//is Palindrome ? true
```

Number Methods

Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects). But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

var x=9.656

Method	How to Use Method	Description	Result

toString()	var x=123;
	var y=x.toString();
	console.log(x+" "+typeof x);
	console.log(y+" "+typeof y);
		toString() method is used to Convert from Number format to String format.	123 number
			123 string

toExponential()	x.toExponential(2);
	x.toExponential(4);
	x.toExponential(6);
		toExponential() returns a string, with a number rounded and written using exponential notation.
		A parameter defines the number of characters behind the decimal point.
		The parameter is optional. If you don't specify it, JavaScript will not round the number.
			9.66e+0
			9.6560e+0
			9.656000e+0

toFixed()	x.toFixed(0);
	x.toFixed(1);
	x.toFixed(4);
	x.toFixed(6);
		toFixed() returns a string, with the number written with a specified number of decimals.
		toFixed(2) is perfect for working with money.
			10
			9.7
			9.6560
			9.656000

toPrecision()	x.toPrecision();
	x.toPrecision(2);
	x.toPrecision(4);
	x.toPrecision(6);
		toPrecision() returns a string, with a number written with a specified length	9.656
			9.7
			9.656
			9.65600

Ex

```
//Number Methods
//toString() used to Convert number to String
var num1=9.656;
var num2=num1.toString();
console.log(num1+" type is : "+typeof num1);    //9.656 type is number
console.log(num2+" type is : "+typeof num2);    //9.656 type is string

//toExponential() Method
var x=9.656;
console.log(x.toExponential(2)); //9.66e+0
console.log(x.toExponential(4)); //9.6560e+0
console.log(x.toExponential(6)); //9.656000e+0
```

```
//toFixed() Method
x=9.656;
console.log(x.toFixed(0)); //10
console.log(x.toFixed(1)); //9.7
console.log(x.toFixed(4)); //9.6560
console.log(x.toFixed(6)); //9.656000

//toPrecision() Method
x=9.656;
console.log(x.toPrecision());  //9.656
console.log(x.toPrecision(2)); //9.7
console.log(x.toPrecision(4)); //9.656
console.log(x.toPrecision(6)); //9.65600
```

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers, those are Number, parseInt and parseFloat.

	Number	parseInt	parseFloat

var con=true;
console.log(Number(con));
console.log(parseInt(con));
console.log(parseFloat(con));
	1	NAN	NAN

var con=false	0	NAN	NAN
var con=”10”;	10	10	10
var con=” 10”;	10	10	10
var con=”10 ”;	10	10	10
var con=” 10 ”;	10	10	10
var con=”10.33”;	10.33	10	10.33
var con=”10,33”;	NAN	10	10
var con=”10 33”;	NAN	10	10
var con=”John”;	NAN	NAN	NAN
var con=”10 John”;	NAN	10	10

Number Properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
POSITIVE_INFINITY	Represents infinity (returned on overflow)
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value

Ex

```
/* ---------------------------------------------------------------------
                            Number Object Examples
 --------------------------------------------------------------------- */

var min = Number.MIN_VALUE;
console.log(min);   //5e-324

var max = Number.MAX_VALUE;
console.log(max);   //1.7976931348623157e+308

var positiveInfinity = Number.POSITIVE_INFINITY;
console.log(positiveInfinity);  //Infinity

var negInfinity = Number.NEGATIVE_INFINITY;
console.log(negInfinity);   //-Infinity

var str = "120";
var num = parseInt(str);
console.log('value : ' + num + ' type : ' + typeof num);
//value : 120 type : number

var newStr = num.toString();
console.log('value : ' + newStr + ' type : ' + typeof newStr);
//value : 120 type : string
```

JavaScript Operators:

- The Operators are used to perform different kind of Operations like conditional or arithmetic using the existing variable data.
- JavaScript supports the following operators.
    - Assignment operator (=)
    - Arithmetic operators (+,-,*,/)
    - Short hand Math Operator (+=,-=,*=,/=)
    - Conditional Operators (<,>,<=,>=)
    - Unary Operator(++,--)
    - Logical Operator(AND,OR)
    - String Concatenation OperatorTernary Operator(? :)
    - = = (type coercision) Operator
    - = = = Operator

Assignment operator =

- We use this operator to assign any value to a variable.

Ex

```
// Assignment operator =
var jsVersion = 'ES5';
console.log('jsVersion is : ' + jsVersion);
//jsVersion is : ES5
```

Arithmetic operators:

- We use this operators we can perform some arithmetic operations.

Ex

```
// Arithmetic operators + - * /
var num1 = 10;
var num2 = 20;
var sum = num1 + num2;
console.log('The Sum of ' + num1 + ' , ' + num2 + ' is : ' + sum);
//The Sum of 10,20 is : 30
```

Short hand Math Operator (+=,-=,*=,/=)

- These operators are used as a shortcut for some arithmetic operations.

Ex

```
// Short hand math += , -= , *= , /=
var a = 10;
var b = 20;
var add = 0;
add = add + (a + b);
add += a + b;
console.log('The add value is  : ' + add);
//The add value is  : 60
```

Conditional Operators (<,>,<=,>=):

- These Operators is used to check the condition between two variables.

Ex

```
// Conditional operators < , > , <= , >= , !==
var age = 25;
if(age < 18){
    console.log('You are Minor');
}
else{
    console.log('You are Major');
}
```

Unary Operator(++,--)

- These operator is used to increment or decrement the value by one.

Ex

```
// Unary Operator ++ , -- (pre , post)
var i = 10;
i = i + 1;
i += 1;
i++;
console.log('i value is : ' + i); // 13
```

Logical Operator(AND,OR)

- These operators is used to evaluate the variables by applying the logical AND or OR operations.
  The AND Operator evaluate to true if both the sides are true.
  The OR Operator evaluate to false if any one side is false.

Ex

```
T && T  >> true;
T && F  >> false;
F && T  >> false;
F && F  >> false;
```

```
T || T  >> true;
F || T  >> true;
T || F  >> true;
F || F  >> false;
```

Ex

```
// Logical operators AND , OR
var inRelation = true;
var parentsAgreed = false;
if(inRelation && parentsAgreed){
    console.log('Get Marry Soon');
}
else{
    console.log('Wait until parents Agreed');
}
```

String Concatenation Operator (+)

- The + operator acts as the string concatenation as well as addition operator.
- If both the operands is number then the plus operator acts as addition.

Ex

```
var k=10+20+30;
console.log(k);    //60
```

- If either side of the plus symbol is a string then its acts as string concatenation operator.

Ex

```
var k=10+20+"30";
var j="10"+20+30;
console.log(k+"--"+j);    //3030--102030
```

We normally use this string concatenation operator to append any dynamic value to a static string.

Ex

```
// String Concatenation Operator
var str = 10 + 20 + "10" + 20 + 10;
console.log('Str value is : ' + str);  //30102010
```

Ternary Operator (? :)

- This operator is used as a shortcut to if-else condition.

Syn: (condition)?(true):(false)

Ex

```
// Ternary operator (? :)
var age=40;
(age > 18) ? console.log('You are Major') : console.log('You are Minor');
//You are Major
```

= = Operator (type coercision):

- This Operator is used to check the equality between two Operands. It checks the data but not datatypes.
- If use == double equals to check the equality the JavaScript performs the following thing:
    - Automatic type coercision or type conversion
    - Actual Comparision of two variables.

Ex

```
// == operator
var x = 10;
var y = "10";
if(x == y){
    console.log('Both are Equal');
}
else{
    console.log('Both are NOT Equal');
}
//Both are Equal
```

= = = Operator:

- === is called as strict equality operator which returns true when the two operands are having the same value without any type conversion.
- This is also used to check the equality between two operands but here no automatic type coercision happens and it directly compares the two variables.

Ex

```
// === operator
x = 10;
y = "10";
if(x === y){
    console.log('Both are Equal');
}
else{
    console.log('Both are NOT Equal');
}
//Both are NOT Equal
```

//it is false because **a** is number and **b** is string.

Note: In JavaScript we should always use === operator to check the comparison between two operands. === checks the data and data type both.

What is the difference between (= = =) and type coercion or double equals (= =)?

Type coercion is the process of converting value from one type to another (such as string to number, object to Boolean, and so on). So, internal type conversion happens here. For Example variable a contains number 30 and variable b contains string '30', let's compare the two values with double equals (= =) then we will get the unexpected result as ***true.***

- Here what happens internally the variable type of b is string, and other one is integer, then the variable type of b is converted into integer then both values are true.

```
EX:
var a=30; //Integer
var b='30'; //String
if(a==b){
  console.log("this is true")
}
else{
  console.log("this is false")
}

Output: this is true
```

- For exact comparisons JavaScript programs were introduced tribal equals (= = =). The tribal equal's compares the type of the variable and it will not converted to another type. This time we will get the expected result. It compares both value and type.

```
EX:
var a=30; //Integer
var b='30'; //String
if(a===b){
  console.log("this is true")
}
else{
  console.log("this is false")
}
Output: this is false
```

Note: Converting a value from one type to another is often called "type casting," when done explicitly, and "coercion" when done implicitly.

The typeof Operator

- typeof operator is used to check the type of data, what datatype of the variable is holds.
- The typeof operator does not return object for functions.
- The **typeof** operator returns the type of a variable or an expression:

```
typeof 0                    // Returns "number"
typeof 314                  // Returns "number"
typeof 3.14                 // Returns "number"
typeof (3)                  // Returns "number"
typeof (3 + 4)              // Returns "number"

typeof ""                   // Returns "string"
typeof "John"               // Returns "string"
typeof "John Doe"           // Returns "string"

var car;                  // Value is undefined, type is undefined

car = undefined;          // Value is undefined, type is undefined

typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined"
typeof null                   // Returns "object"
```

- The typeof operator returns object for both arrays, and null.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;        // Now value is null, but type is still an object
```

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined;   // Now both value and type is undefined
```

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

Note: In case of JavaScript there is no way of knowing what is the type of variable is holds at that time. It's important to find out the type of variables what type of variable it is. We have to figure out in dynamically.

Working with boolean

- For **integers** all non-zero values it will return true either it may be positive or negative value. The value of zero it will return false.

```
var a=0; //zero value
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:false
```

```
var a=20; //Non-zero value
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:true
```

- For Strings all the non-empty strings it will return true, for empty string it will return false.

```
var a="Delhi"; //Non-empty string
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:true
```

```
var a=""; //empty string
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:false
```

- For undefined and null values always return false.

```
var a; //undefined
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:false
```

```
var a=null; //defined as null
if(a){
  console.log("true")
}
else{
  console.log("false")
}

Output:false
```

Conditional Statements:

- The Conditional statements are used to evaluate various conditions and apply looping based on the conditions.
- In JavaScript Conditional Statements are
  - if else
  - switch
- In JavaScript looping statement depending on the condition
  - for loop
  - while loop
  - do while loop

if else statement:

- The if else statement is used to check the condition between two Operands and if the condition is true it executes the if block, if the condition is false it executes else block.

Syn:

```
if(condition){
    //true part
} else{
    //false part
}
```

In the above Syntax it applicable for only one condition if you wants to check two or more conditions we have to add else-if block for each extra condition.

Syn:

```
if(condition){
    //true part
} else if(condition){
    // true part
} else{
    //false part
}
```

Ex_1:

```
// If Else condition Example
var courseCompleted = true;
var practiceCompleted = false;
if(courseCompleted && practiceCompleted){
    console.log('You will get the Job Soon');
}
else if(courseCompleted && !practiceCompleted){
    console.log('Please start practicing');
}
else{
    console.log('Please join any course');
}
//Please start practicing
```

Ex_2:

```
//Example 2
var time=20;
if(time<=12){
    console.log("Good Morning ");
}
```

```
else if(time>12 && time<=17){
    console.log("Good Afternoon");
}
else if(time>17 && time<=23){
    console.log("Good Evening ");
}
else{
    console.log("Enter Currect time");
}
//Good Evening
```

Note: By using if else Condition to check more no. of conditions it starts from condition 1 and all the way to the actual condition. this increases the execution time of the program.
To improve the performance of checking these conditions we should use switch statement.

Switch Statement:

- The Switch statement is alternate for if-else condition.
- If there are multiple conditions the switch statement is better than if-else statement.

Syn:

```
switch(condition){
    case 0:
        //Statements
        break;
case 1:
        //Statements
        break;
default:
        //Statements
        break;
}
```

- If else, switch statement must contain the default section, this is same as the else condition of if-else statement. In the switch statement if the no case is match then it executes the ***default*** statement.
- When JavaScript reaches a **break** keyword, it breaks out of the switch block.
- When matching case is found then it stop the execution from the switch case. There is no need for testing remaining cases.
- A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.
- It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Ex

```
// Switch Statement Example
var today = new Date().getDay();
output = 'Today is : ';
switch(today){
    case 0:
        output += ' Sunday ';
        break;
    case 1:
        output += ' Monday ';
        break;
    case 2:
        output += ' Tuesday ';
        break;
```

```
            case 3:
                output += ' Wednesday ';
                break;
            case 4:
                output += ' Thursday ';
                break;
            case 5:
                output += ' Friday ';
                break;
            case 6:
                output += ' Saturday ';
                break;
            default:
                output += ' ';
                break;
        }
        console.log(output);
        document.querySelector('#display').textContent = output;
```

Print the numberString

```
        // Print the numberString
        var numberString = "787";
        var tempString = '';
        for(var i = 0; i<numberString.length; i++){
            var ch = parseInt(numberString.charAt(i));
            switch(ch){
                case 0:
                    tempString += ' ZERO ';
                    break;
                case 1:
                    tempString += ' ONE ';
                    break;
                case 2:
                    tempString += ' TWO ';
                    break;
                case 3:
                    tempString += ' THREE ';
                    break;
                case 4:
                    tempString += ' FOUR ';
                    break;
                case 5:
                    tempString += ' FIVE ';
                    break;
                case 6:
                    tempString += ' SIX ';
                    break;
                case 7:
                    tempString += ' SEVEN ';
                    break;
                case 8:
                    tempString += ' EIGHT ';
                    break;
                case 9:
                    tempString += ' NINE ';
                    break;
```

```
            default:
                tempString += ' ';
                break;
        }
    }
    console.log(tempString);
    document.querySelector('#display').textContent = tempString;
    //SEVEN EIGHT SEVEN
```

for loop

- This statement is used to loop the variables with respective to condition.

Syn:

```
for(intialization;condition;increament/decrement){
    //Statements
}
```

- for loop contains three parts such as intialization,condition and increament/decrement section.
- In for loop the intialization happens only once and until the condition is true it keep executing the for loop, If the condition is fail then only the control comes outside the for loop.

Ex_1:

```
// For loop Example to display from 1 - 10 values
var output = '';
for(var i = 1; i<= 10; i++){
    if(i <= 9){
        output += i + " , ";
    }
    else{
        output += i;
    }
}
console.log(output);
document.querySelector('#display').textContent = output;
```

Ex_2: *if you are using break inside the condition it will stops the execution.*

```
//for and break
for(var y=0;y<5;y++){
    if(y === 2){
        break;
    }
    console.log(y);
}
```

Output:

```
0
1
```

Ex_3: *if you are using continue keyword inside the condition it will continue the execution and skip what the statement what condition is satisified.* *Result:*

```
//for and continue
for(var k=0;k<5;k++){
    if(k === 2){
        continue;
    }
    console.log(k);
}
```

```
0
1
3
4
```

for in loop

The JavaScript for/in statement loops through the properties of an object.

```
//For in loop
var cars_details={"carColor":"black","carPrice":500000,"carType":"petrol"};
for(var i in cars_details){
    console.log(cars_details[i]);
}
```

Output:

```
black
500000
petrol
```

forEach loop:

```
//forEach loop on arrays
var fruits=["Banana","Orange","Graps","Apple"];
fruits.forEach(function (element) {
    console.log(element);
});
```

Output:

```
Banana
Orange
Graps
Apple
```

while loop:

- This Statement is also used to loop through a condition.

Syn:

```
intialization
while(condition){
    //statements
    increament/decrement
}
```

- In the while loop first its checks the condition once the condition is true then only the control will jump inside the value.
- If the Condition is fail then the control will jump to outside the while loop.
- To Print the values from 1 to 10 using while loop is as follows:

While loop Example to display from 1 - 10 values

```
// While loop Example to display from 1 - 10 values
output = '';
// initialization
i = 1;
// Condition
while(i <= 10){
    // statements
    if(i <= 9){
        output += i + " - ";
    }
    else{
        output += i;
    }
```

```
    // increment / decrement
    i++;
}
console.log(output);
document.querySelector('#display').textContent = output;
```

Result:

```
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
```

Note: In this while loop the Increment/Decrement Section is Mandatory. If you don't add increment/decrement section then the control will goes to infinite loop.

do-while loop:

- This is also used to loop through condition, it is almost same as while loop but in while loop until unless the condition is successful the control will not jump inside the while loop but in do while loop atleast once it executes the statements and then checks the condition.

Syn:

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

Do while loop Example to display from 1 - 10 values

```
// Do while loop Example to display from 1 - 10 values
output = '';
i = 1;
do{
    // statements
    if(i <= 9){
        output += i + " * ";
    }
    else{
        output += i;
    }
    // increment / decrement
    i++;
}

while(i <= 10);
console.log(output);
document.querySelector('#display').textContent = output;
```

Result:

```
1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10
```

Object

- Instead of creating the individual variables we can group all the individual variables into a single variable called as object. An **object** is a collection of properties and methods. An object can be representing with curly braces.
- A property is associated with a name property name (key) and value.
- Property name and value separated by a colon.
- Objects are variables too, but objects can contain many values.

Ex

```
//Create an Object in JavaScript
    var employee={};

//add properties to an object
    employee.name="John";
    employee.age=40;
    employee.designation="Manager";

//access an Object
    console.log(employee);
```

Result:

```
▼{name: "John", age: 40, designation: "Manager"}
   age: 40
   designation: "Manager"
   name: "John"
```

Object literal:

- A JavaScript ***object literal*** is a **name (or key): value** pairs (name and value separated by a colon) and wrapped in curly braces. A comma separates each name-value pair from the next.
- There should be no comma after the last name-value pair.

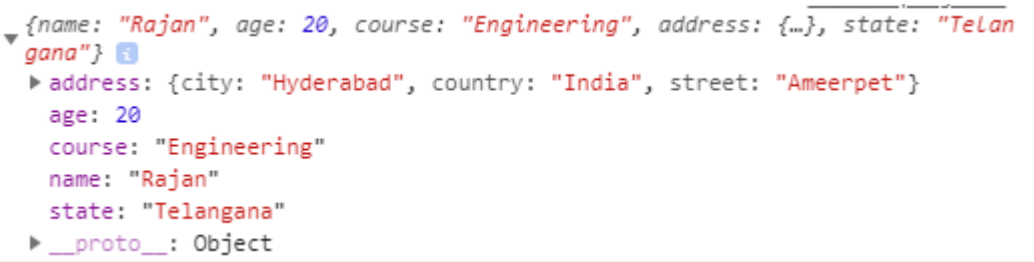
The following demonstrates an example object literal:

```
//Create Object literal
    var mobile={
        brand:'Lenovo',
        color:'black',
        RAM:'2GB',
        storage:'32GB',
        hasInsurance:false
    };
    console.log(mobile);
```

- ***Object literal property*** values can be of any data type, including array literals, functions, and nested object literals. Here is another object literal example with these property types:

```
var student={
    name:'Rajan',
    age:20,
    course:'Engineering',
    address:{
        city:'Hyderabad',
        country:'India',
    },
    state:state()
};
function state() {
    return "Telangana";
}
console.log(student);
```

Result:



- We can even access the properties of an object using bracket notation also as follows:

Ex:

```
console.log(student.name);     //Rajan, dot notation
console.log(student['name']);   //Rajan, [ ] notation
```

- If you are trying to access the property which is not exists in the object then we will get the value undefined.

Ex:

```
console.log(student.collegename); //undefined
```

- Create an Employee object which is contains firstName, lastName, gender and designation. This one is repeate for every employee. Like this happens in below Code

```
var emp1 = {};
emp1.firstName = "Michael";
emp1.lastName = "Scott";
emp1.gender = "M";
emp1.designation = "Regional Manager";

var emp2 = {};
emp2.firstName = "Dwight";
emp2.lastName = "Schrute";
emp2.gender = "M";
emp2.designation = "Assistant to the Regional Manager";
```

- For example in the company totally 200 members are working, for every employee we need to create 200 objects for employees. Then it's very difficult to create, it takes lot of time to develop and code redundancy.

- When a piece of code repeats multiple times then create a function and pass the data as argument.

```
function createEmployeeObject(firstName, lastName, gender, designation) {
    var newObject = {};
    newObject.firstName = firstName;
    newObject.lastName = lastName;
    newObject.gender = gender;
    newObject.designation = designation;
    return newObject;
}
```

- Now we can create an 3rd employee details, and pass the employee values to the function "createEmployeeObject".

```
var emp3 = createEmployeeObject("Jim", "Halpert", "M", "Sales Representative");
```

- Now we can access three employee objects in console.

```
emp1
Object { firstName: "Michael", lastName: "Scott", gender: "M", designation: "Regional
Manager" }
emp2
Object { firstName: "Dwight", lastName: "Schrute", gender: "M", designation: "Assistant to
the Regional Manager" }
emp3
Object { firstName: "Jim", lastName: "Halpert", gender: "M", designation: "Sales
Representative" }
```

- Now emp3 also getting same as previous employees. If repeated code is available then we can create a function and pass the values as arguments to the function. So, it saves the lot of time & reduce the code redundancy.

How many Ways to create an Object?

1. *Using Object constructor*

```
var car=new Object();
car.color="orange";
car.make="Skoda";
console.log(car);
```

Output:

```
▶ Object { color: "orange", make: "Skoda" }
```

2. *Using Object.create() method:*

```
var person=Object.create(null);
person.firstname="satish";
person.age=90;
console.log(person);
```

Output:

```
▶ Object { firstname: "satish", age: 90 }
```

3. *Using brackets {}*

- This method creates a new object extending the prototype object passed as a parameter.
- This is equivalent to Object.create(null) method, using a null prototype as an argument.

```
var person={};
person.name="satish andra";
person.age=32;
person.location="samalkot";
console.log(person);
```

Output:

```
Object { name: "satish andra", age: 32, location: "samalkot" }
```

- 3rd one is the best approach while creating an object. 1st and 2nd not recommended it gives more complexity.

How can you access the properties of an object?

- There are two ways to access the properties of an object.
  - Dot Notation (.)
  - Bracket Notation([ ])

Dot notation:

- Property identifies can only be alphanumeric (and _ and $)

```
var person={f_name:"Satish",lname:"Andra"};
console.log(person.f_name);
```

Output:

```
Satish
```

- Identifier or property name shouldn't starts with digits or should not contain variables then it will throw an error.

```
var person={1fname:"Satish",lname:"Andra"};
console.log(person.1fname);

Output:
/*
Exception: SyntaxError: identifier starts immediately after numeric literal
@Scratchpad/1:1
*/
```

Bracket notation:

- When working with bracket notation, property identifiers only have to be a String.
- They can include any characters, including spaces. Variables may also be used as long as the variable resolves to a String.

```
var person={"1fname":"Sudheer","lname":"Andra"};
console.log(person["1fname"]);
```

Output:

```
Sudheer
```

Nested Object:

- Declaring an object inside the multiple objects is nothing but a nested object.

```
const user = {
    id: 101,
    email: 'jack@dev.com',
    personalInfo: {
        name: 'Jack',
        address: {
            line1: 'westwish st',
            line2: 'washmasher',
            city: 'wallas',
            state: 'WX'
        }
    }
}
console.log(user.personalInfo.address);
```

Output:

```
▶ Object { line1: "westwish st", line2: "washmasher", city: "wallas", state: "WX" }
```

= = = for Objects:

- = = = always compares with object reference if object references are equal then it will return true otherwise it will return's false.
- In the below example demonstrates that the both references are equal. Both objects pointing to the single address.

```
var person1={"firstname":"satish","lastname":"andra"};
var person2=person1;
if(person1===person2){
    console.log("Both objects are equal");
    console.log(person2.firstname);
}
else{
    console.log("Both objects are not equal");
}
Output:
Both objects are equal
satish
```

- Let us see the below example demonstrates that, Here two objects are created both are pointing to different address because of this reason the else block executed at below.

```
var person1={"firstname":"satish","lastname":"andra"};
var person2={"firstname":"satish","lastname":"andra"};
if(person1===person2){
    console.log("Both objects are equal");
    console.log(person2.firstname);
}
else{
    console.log("Both objects are not equal");
}
Output:
    Both objects are not equal
```

Why and How We Use Object Literals

- No need to invoke constructors directly or maintain the correct order of arguments passed to functions.
- Object literals are also useful for event handling; they can hold the data otherwise be passed in function calls from HTML event handler attributes.

Revisiting undefined and null

- In JavaScript if you want to access the undefined property name of an object it will not thrown any error. You will get the result as undefined.

```
var person={"firstname":"satish","lastname":"andra"};
console.log(person.middlename);

output:
undefined
```

- In the above example you did not define the middle name. The middle name is optional for some persons. In this situation we will define the middle name as null.
- If the person not having the middle name this time we will get the null.

```
var person={"firstname":"satish","lastname":"andra","middlename":null};
console.log(person.middlename);
```

Output: null

Deleting Properties with delete operator

```
var person = {
   'firstName': 'Koushik',
   'middleName': null,
   'lastName': 'Kothagal',
   'age': 25
};
...

person.age = undefined;
```

- If you don't want property age then we will use object.property name as *undefined*, if you access whole object still we will get four property names but age property is not deleted from the object. Still it is existed in the object.

```
person = {
    'firstName': 'Koushik',
    'middleName': null,
    'lastName': 'Kothagal',
    'age': undefined
}
```

- If you are accessing the specific age property then you will get the same result as *undefined* in the both cases if property is existed or not in the object.
- See the below examples this time both will get the same result as *undefined*. So, no problem in this case.

```
var person={
  "firstname":"satish",
  "lastname":"andra",
  "middlename":null,
};
console.log(person.age);
```

```
var person={
   "firstname":"satish",
   "lastname":"andra",
   "middlename":null,
   "age":23
};
person.age=undefined;
console.log(person.age);
```

- If you access the whole objects then this age property still present in the object. It will not delete from the object.

```
var person={
   "firstname":"satish",
   "lastname":"andra",
   "middlename":null,
   "age":23
};
person.age=undefined;
console.log(person);
```

Result:

```
▶ Object { firstname: "satish", lastname: "andra", middlename: null, age: undefined }
```

- If you delete the specific property then we will use delete operator, then, it will delete from the object.

```
var person={
   "firstname":"satish",
   "lastname":"andra",
   "middlename":null,
   "age":23
};
delete person.age;
console.log(person);
```

Result:

```
▶ Object { firstname: "satish", lastname: "andra", middlename: null }
```

Arrays

- Arrays are used to store multiple values in a single variable and you can access the values by referring to an index number.
- Array index must be starts from zero (0). An array can be defined in inline using open, close square brackets, individual values in the array separated with comma (,).

```
var myArray = [100, 200, 300];
```

- If you access individual values in array then we use object name with array index values, these index values must be in square brackets.

```
myArray[0];
myArray[1];
myArray[2];
myArray[3];
```

- Here we will access three indexes that means myArray[0] to myArray[2]. If you access the array index *3* then we will get the result as *undefined* like objects. Arrays are an object. In objects also we will get the same result if property name is not present.

```
var even_num=[2,4,6,8];
console.log("even_num[0] is "+even_num[0]);
console.log("even_num[1] is "+even_num[1]);
console.log("even_num[2] is "+even_num[2]);
console.log("even_num[3] is "+even_num[3]);
console.log("even_num[4] is "+even_num[4]);
```

Result:

```
even_num[0] is 2
even_num[1] is 4
even_num[2] is 6
even_num[3] is 8
even_num[4] is undefined
```

- If you want to add the array then we will use object name with an array index Array index must be inside the square bracket, and assign the value.

```
var even_num=[2,4,6,8];
even_num[4]=10;
console.log("even_num[0] is "+even_num[0]);
console.log("even_num[1] is "+even_num[1]);
console.log("even_num[2] is "+even_num[2]);
console.log("even_num[3] is "+even_num[3]);
console.log("even_num[4] is "+even_num[4]);
```

Result:

```
even_num[0] is 2
even_num[1] is 4
even_num[2] is 6
even_num[3] is 8
even_num[4] is 10
```

- Arrays are object that means we are assigning a array to another array and access the first array values.

```
var person_names=["Satish","Sudheer","Mohan","Malleswari"];
var person_names2=person_names;
console.log(person_names2[2]);
```

Result: Mohan

- If you want to find the length of the array then we will use the property called length property. Depending on the array index we will get the length.

```
var even_num=[2,4,6,8];
even_num[400]=10;
even_num[4]=12;
console.log("even_num[0] is "+even_num[0]);
console.log("even_num[400] is "+even_num[400]);
console.log("even_num[4] is "+even_num[4]);

console.log(even_num);
```

Result:

```
even_num[0] is 2
even_num[400] is 10
even_num[4] is 12
▼ (401) […]
  ▶ [0…99]
  ▶ [100…199]
  ▼ [200…299]
  ▶ [300…399]
  ▶ [400…400]
    length: 401
  ▶ <prototype>: Array []
```

- We can't access the array indexes through dot notation because array index can be represented in digits. If you access any property names using dot notation then that property may contains characters, underscore or dollar. Through dot notation we can't access property names as digits.
- Because of this reason we can access the array elements with square brackets only.

```
var person$names=["Satish","Sudheer"];
console.log(person$names[0]);
console.log(person$names["1"]);
```

Result:

```
Satish
Sudheer
```

Here in the above example, we are accessing first array element with array index with *number zero (0)*, and next statement we are accessing the second array element with array index with *string value "1".*

How it will get and how to get result, Here internal type conversion was happened, the first array index number will be converted into string, like this we will get the array element.

Ex_1: *Program on how to get first and last array element.*

```
var fruits=['Banana','Orange','Lemon','Apple','Kiwi'];

//Accessing First array Element
console.log('Accessing first array Element is '+fruits[0]);

//Accessing Last array Element
var last=fruits[fruits.length-1];
console.log('Accessing Last array Element is '+last);
```

Result:

Accessing first array Element is Banana
Accessing Last array Element is kiwi

Ex_2: *Program on how to concatenate two or more arrays.*

```
var studentDetails=['ZPHigh','Dulla'];
var studentInfo=['Satish','Class10th'];
var studentAddress=['Dulla','Kadiam Mandal'];
var student=studentDetails.concat(studentInfo,studentAddress);
console.log(student);
```

Result:

```
0: "ZPHigh"
1: "Dulla"
2: "Satish"
3: "Class10th"
4: "Dulla"
5: "Kadiam Mandal"
length: 6
```

Associative Arrays

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does **not** support arrays with named indexes.
- In JavaScript, **arrays** always use **numbered indexes**.

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;          // person.length will return 0
var y = person[0];              // person[0] will return undefined
```

***Note*:** If you use named indexes, JavaScript will redefine the array to a standard object. After that, some array methods and properties will produce **incorrect results**.

The Difference between Arrays and Objects

- In JavaScript, **arrays** use **numbered indexes**.
- In JavaScript, **objects** use **named indexes**.
- Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays & When to use Objects.

1. JavaScript does not support associative arrays.
2. You should use **objects** when you want the element names to be **strings (text)**.
3. You should use **arrays** when you want the element names to be **numbers**.

Array Methods:

- The JavaScript array contains the predefined methods or functions:

Original Array		
fruits=['Banana','Orange','Apple','Graps']		
Method	**Description**	**Result**
fruits.join(‘ * ‘);	Join the array elements with what you want	Banana * Orange * Apple * Graps
fruits.push('Lemon');	Adding the element at End of the array	Banana,Orange,Apple,Graps,Lemon
fruits.pop();	Remove the Array element from the last	Banana,Orange,Apple
fruits.shift();	Remove the Array element from the first	Orange,Apple,Graps
fruits.unshift('Lemon');	Adding the element at first of the array	Lemon,Banana,Orange,Apple,Graps
fruits.splice(2);	Remove index values after index 2	Banana, Orange
fruits.splice(2, 0, "Lemon", "Kiwi");	Here 1st parameter represents where the new element should be added , 2nd parameter represents how many elements removes from the array	Banana,Orange,Lemon,Kiwi,Apple,Graps
fruits.splice(0,1);		Orange,Apple,Graps
fruits.slice(1);	Here one parameter represents to remove element from array	Orange,Apple,Graps
fruits.slice(3);	Removes first three values	Graps
fruits.slice(1,3);	2nd parameter counting the no.of elements in array; counting from 1st element i.e,, Banana,Orange, Apple and 1st parameter indicates how many elements removed from the array what you got.	Orange,Apple
fruits.slice(2,4);		Apple,Graps
fruits.sort();	Sort in Ascending Order	Apple,Banana,Graps,Orange
fruits.reverse();	Reverse Order of an Array	Graps,Apple,Orange,Banana

Ex:

```
var fruits=['Banana','Orange','Apple','Graps'];
var join=fruits.join(' * ');
console.log("The Joining Elements of an Array is "+join);

fruits.push('Lemon');
console.log('The pushing element added at last '+fruits);

fruits.pop();
console.log('The popping element removed from last '+fruits);

fruits.unshift('Lemon');
console.log('The unshift element added at first '+fruits);

fruits.shift();
console.log('The shift element removed from first '+fruits);

fruits.splice(2,0,'Kiwi','Dragon','Mango');
console.log("The added elements from 3rd Position "+fruits);

fruits.splice(3,1,'Coco');
console.log("Remove one element and add Coco at 3rd Position "+fruits);

fruits=['Banana','Orange','Apple','Graps'];
console.log(fruits.slice(1));

fruits=['Banana','Orange','Apple','Graps'];
console.log(fruits.slice(2));

fruits=['Banana','Orange','Apple','Graps'];
console.log(fruits.slice(2,3));

fruits=['Banana','Orange','Apple','Graps'];
console.log(fruits.sort());

fruits=['Banana','Orange','Apple','Graps'];
console.log(fruits.reverse());
```

Result:

```
The Joining Elements of an Array is Banana * Orange * Apple * Graps
The pushing element added at last Banana,Orange,Apple,Graps,Lemon
The popping element removed from last Banana,Orange,Apple,Graps
The unshift element added at first Lemon,Banana,Orange,Apple,Graps
The shift element removed from first Banana,Orange,Apple,Graps
The added elements from 3rd Position Banana,Orange,Kiwi,Dragon,Mango,Apple,Graps
Remove one element and add Coco at 3rd Position Banana,Orange,Kiwi,Coco,Mango,Apple,Graps
▶ Array(3) [ "Orange", "Apple", "Graps" ]
▶ Array [ "Apple", "Graps" ]
▶ Array [ "Apple" ]
▶ Array(4) [ "Apple", "Banana", "Graps", "Orange" ]
▶ Array(4) [ "Graps", "Apple", "Orange", "Banana" ]
```

indexOf(value)

- In JavaScript array the index of function is used to find the index of the provided value of an array.
- If we are trying to find the index of the Non-existed value of an array we get the value -1.

Ex:

```
var array=[10,20,30,40];
var names=['satish','sudheer','mohan']
console.log(array.indexOf(20));         //1
console.log(array.indexOf(90));         //-1
console.log(names.indexOf('sudheer'));  //1
console.log(names.indexOf('Sudheer'));  //-1
```

Deleting Elements

JavaScript arrays are objects; elements can be deleted by using the JavaScript operator **delete**:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];           // Changes the first element in fruits to undefined
```

Note: Using **delete** may leave undefined holes in the array. Use pop() or shift() instead.

Ex:

```
var names=['satish','sudheer','mohan']
delete names[0];
console.log(names);
```

Result:

```
Array(3) [ <1 empty slot>, "sudheer", "mohan" ]
```

Indexes 1:'sudheer'
2:'mohan'

Numeric Sort

- By default, the sort() function sorts values as **strings**. This works well for strings ("Apple" comes before "Banana").
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1". Because of this, the sort() method will produce incorrect result when sorting numbers.

Ex

```
var rollno=[60,10,321,20,2,23423,232];
rollno.sort();
console.log(rollno);
```

Result:

```
Array(7) [ 10, 2, 20, 232, 23423, 321, 60 ]
```

*You can fix this by providing a **compare function**:*

For Ascending Order:

```
var rollno=[60,10,321,20,2,23423,232];
rollno.sort(function(a,b){
  return a-b;
});
console.log(rollno);
```

Result:

```
Array(7) [ 2, 10, 20, 60, 232, 321, 23423 ]
```

For Descending Order:

```
var rollno=[60,10,321,20,2,23423,232];
rollno.sort(function(a,b){
  return b-a;
});
console.log(rollno);
```

Result:

```
Array(7) [ 23423, 321, 232, 60, 20, 10, 2 ]
```

Sort the numbers in alphabetically and numerically using buttons.

```
<button onclick="alphabetically()">sort alphabetic</button>
<button onclick="numerically()">sort numerically</button>
<p id="click"></p>
<script>
    var numbers=[10,342,21,34,234,3534,12,3];
    document.getElementById('click').innerHTML=numbers;
    function alphabetically() {
        numbers.sort();
        document.getElementById('click').innerHTML=numbers;
    }
    function numerically() {
        numbers.sort(function (a,b) {
            return a-b;
        });
        document.getElementById('click').innerHTML=numbers;
    }
</script>
```

Result:

sort alphabetic | sort numerically

3,10,12,21,34,234,342,3534

How to Recognize an Array

- The JavaScript operator **typeof** returns "object":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits;              // returns object
```

- The typeof operator returns object because a JavaScript array is an object.
- To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:

```
console.log(Array.isArray(names));
```

Array Iterations:

Method	Description
array.map()	Connecting the relationship between arrays
array.filter	Conditional usage
array.every()	Checks all the elements; all elements matches with condition then only it will return true
array.some()	Checks all the elements; some of the elements matches with condition then it will return true.
array.find()	Processed first array element
array.findIndex()	Processed first array element index
array.reduce()	It will give one element as final output; from left to right its processed by default
array.reduceRight()	Same as reduce() but different is it Process from right to left
array.indexOf()	It will return index of an string
array.lastIndexOf()	It will return last index of an string if more than one string is found
array.forEach()	Iterations like for loop

Iteration Name	Variable Declarion	Function Declaration	Result

map()	var num2=num1.map(myfun);	function myfun(array) {
		return array*12;
		}
			[120, 264, 144, 144,240]

filter()	var age=num1.filter(myfun);	function myfun(index) {
		return index>18;
		}
			22,20

every()	var age=num1.every(myfun);	function myfun(value) {
		return index>18;
		}
			false

some()	var age=num1.some(myfun);	function myfun(value) {
		return value>18;
		}
			true

find()	var age=num1.find(myfun);	function myfun(array) {
		return array>18;
		}
			20

reduce()	var age=num1.reduce(myfun);
	var total=0;
		function myfun(array) {
		return total+array;
		}
			76

forEach()	num1.forEach(myfun);
	var text='';
		function myfun(array) {
		return text+=array;
		}
			10,22,12,12,20

Array.map()

- The map() method creates a new array by performing a function on each array element.
- The map() method does not execute the function for array elements without values.
- The map() method does not change the original array.
- This example multiplies each array value by 2:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array.map()</h2>
<p>Creates a new array by performing a function on each array element.</p>
<p id="demo"></p>

<script>
    var numbers1 = [45, 4, 9, 16, 25];
    var numbers2 = numbers1.map(myFunction);
    document.getElementById("demo").innerHTML = numbers2;
    function myFunction(value, index, array) {
        return value * 2;
    }
</script>

</body>
</html>
```

Result:

JavaScript Array.map()

Creates a new array by performing a function on each array element.
90, 8,18,32,50

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted:

```
<!DOCTYPE html>
<html>
<body>

    <h2>JavaScript Array.map()</h2>
    <p>Creates a new array by performing a function on each array element.</p>
    <p id="demo"></p>

<script>
    var numbers1 = [45, 4, 9, 16, 25];
    var numbers2 = numbers1.map(myFunction);
    document.getElementById("demo").innerHTML = numbers2;
    function myFunction(value) {
        return value * 2;
    }
</script>

</body>
</html>
```

Array.filter()

- The filter() method creates a new array with array elements that passes a test.
- This example creates a new array from elements with a value larger than 18:

In the example below, the callback function does not use the index and array parameters, so they can be omitted:

Array.every()

- The every() method check if all array values pass a test.
- This example check if all array values are larger than 18:

In the example below, When a callback function uses the first parameter only (value), the other parameters can be omitted.

Array.some()

- The some() method check if some array values pass a test.
- This example check if some array values are larger than 18:

In the example below, Array.some() is supported in all browsers except Internet Explorer 8 or earlier.

Array.find()

- The find() method returns the value of the first array element that passes a test function.
- This example finds (returns the value of) the first element that is larger than 18:

In the example below, Array.find() is not supported in older browsers. The first browser versions with full support is listed below.

Array.reduce()

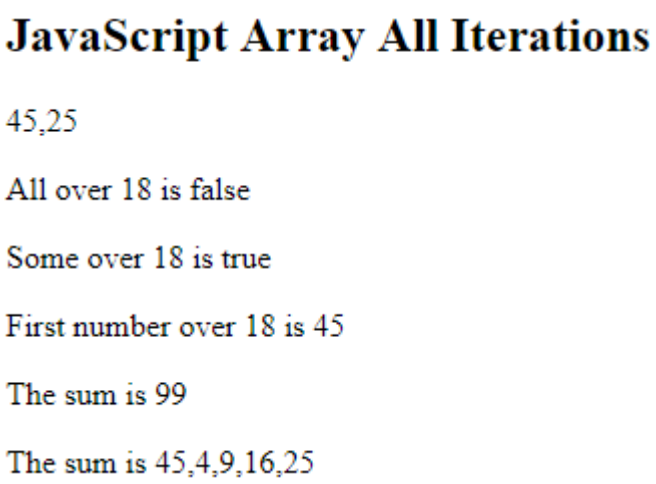
- The reduce() method runs a function on each array element to produce a single value.
- The reduce() method works from left-to-right in the array.
- The reduce() method does not reduce the original array.

The example above does not use the index and array parameters.

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Array All Iterations</h2>
    <p id="filter"></p>
    <p id="every"></p>
    <p id="some"></p>
    <p id="find"></p>
    <p id="reduce"></p>
    <p id="foreach"></p>
```

```
<script>
    var numbers = [45, 4, 9, 16, 25];

    /*-------Array Filter Iteration---------*/
    var over18 = numbers.filter(fun_filter);
    document.getElementById("filter").innerHTML = over18;
    function fun_filter(value, index, array) {
        return value > 18;
    }

    /*-------Array Every Iteration---------*/
    var allOver18 = numbers.every(fun_every);
    document.getElementById("every").innerHTML
        = "All over 18 is " + allOver18;
    function fun_every(value, index, array) {
        return value > 18;
    /*-------Array Some Iteration---------*/
    var someOver18 = numbers.some(fun_some);
    document.getElementById("some").innerHTML
        = "Some over 18 is " + someOver18;
    function fun_some(value, index, array) {
        return value > 18;
    }

    /*-------Array Find Iteration---------*/
    var first = numbers.find(fun_find);
    document.getElementById("find").innerHTML =
        "First number over 18 is " + first;
    function fun_find(value, index, array) {
        return value > 18;
    }
    /*-------Array Reduce Iteration---------*/
    var sum = numbers.reduce(fun_reduce);
    document.getElementById("reduce").innerHTML =
        "The sum is " + sum;
    function fun_reduce(total, value, index, array) {
        return total + value;
    }

    /*-------Array forEach Iteration---------*/
    var txt = "";
    var _numbers = [45, 4, 9, 16, 25];
    _numbers.forEach(fun_forEach);
    document.getElementById("foreach").innerHTML =
        "forEach Values " + _numbers;

    function fun_forEach(value) {
        txt = txt + value + "<br>";
    }
</script>
```

Result:



The Difference between Arrays and Objects

In JavaScript, **arrays** are used for **numbered indexes** and O**bjects** are used for **named indexes**.

Is String is a Primitive?

- Yes, it is a primitive. In the below example demonstrate the typeof(str) returns *string*.

```
var str="Hello World";
console.log(str);
console.log(typeof(str));
console.log(str.length);
```

Result:

```
Hello World
string
11
```

typeof(str) return's the string, string is a primitive but how does dot notation works on primitives it will works on objects?

- JavaScript as an equivalent objects for each of the primitive data types like string object, number object, Boolean object and symbol object.
- Here string is a primitive and as an equivalent string object.
- If str.length should automatically fail if str is a primitive. When you access like this str.length JavaScript does gets that string and converts into the equivalent object, when it converts into the object the length property become available to it and then it calls the length property of the object string and that's how the length works.
- *Here str.length,*

Str is a string then its immediately wraps into wrapper object which is a string object and then .length is a string object gives the length. However, if you are access the typeof(str) again it will get the string because when the object is created the string primitives to string object it is not assigned to the str it is just an temporary objects get its created length property is pulled up from it and object is discarded. Just object is created in fraction of second just make variable.length work. This happens for primitives.

JavaScript Date Object

By default, JavaScript will use the browser's time zone and display a date as a full text string.

Mon Dec 31 2018 11:36:42 GMT+0530 (India Standard Time)

Creating Date Objects

Date objects are created with the **new Date()** constructor.
There are **4 ways** to create a new date object

Date Objects	Result
new Date()	new Date() **creates a new date object with the** current date and time
new Date(*year, month, day, hours, minutes, seconds, milliseconds*)	• new Date(*year, month, ...*) creates a new date object with a specified date and time • 7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order) • JavaScript counts months from 0 to 11. Ex: January is 0. December is 11.
new Date(*milliseconds*)	**new Date(*milliseconds*) creates a new date object as zero time plus milliseconds**
new Date(dateString)	new Date(dateString) creates a new date object from a date string

***Note*:** Date objects are static. The computer time is ticking, but date objects are not

```
// get today's date
var today = new Date();
console.log(today);
//Mon Dec 31 2018 11:50:45 GMT+0530 (India Standard Time)

//pass Manually
var d=new Date(2018,11,25,10,30,59);
console.log(d);
//Tue Dec 25 2018 10:30:59 GMT+0530 (India Standard Time)

//Date in milliSeconds
d=new Date(0);
console.log(d);
//Thu Jan 01 1970 05:30:00 GMT+0530 (India Standard Time)

//DateString
d = new Date("October 13, 2014 11:13:00");
console.log(d);
//Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)
```

JavaScript Get Date Methods

These methods can be used for getting information from a date object

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Ex

```
var _date=new Date();
console.log(_date.getFullYear()); //2018
console.log(_date.getMonth());  //11
console.log(_date.getDate());   //31
console.log(_date.getHours());  //12
console.log(_date.getMinutes()); //7
console.log(_date.getSeconds());    //13
console.log(_date.getMilliseconds());   //551
console.log(_date.getTime());   //1546238259829
console.log(_date.getDay());    //1
```

JavaScript Set Date Methods

Set Date methods are used for setting a part of a date.

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Ex

```
// get full day of the week using switch statement
today = new Date().getDay();
output = 'Today is : ';
switch(today){
    case 0:
        output += ' Sunday ';
        break;
    case 1:
        output += ' Monday ';
        break;
    case 2:
        output += ' Tuesday ';
        break;
    case 3:
        output += ' Wednesday ';
        break;
    case 4:
        output += ' Thursday ';
        break;
    case 5:
        output += ' Friday ';
        break;
    case 6:
        output += ' Saturday ';
        break;
    default:
        output += ' ';
        break;
}
console.log(output);
document.querySelector('#display').textContent = output;
```

Result:

```
Today is :  Monday
```

Exercise_2:

Html File

```
<section class="time-section">
    <!-- Indian Clock -->
    <article class="time-article">
        <h1>India</h1>
        <h3 id="india-date"></h3>
        <h1 id="india-time"></h1>
    </article>

    <!-- USA Clock -->
    <article class="time-article">
        <h1>USA</h1>
        <h3 id="us-date"></h3>
        <h1 id="us-time"></h1>
    </article>
```

CSS File

```
.time-section{
    margin-left: 80px;
}
.time-article{
    width: 380px;
    float: left;
    text-align: center;
    margin: 10px;
```

```
    <!-- China Clock -->
    <article class="time-article">
        <h1>China</h1>
        <h3 id="china-date"></h3>
        <h1 id="china-time"></h1>
    </article>
</section>

<script src="09_script.js"></script>
```

```
    background: white url("../img/lightbulbs.jpg")
                no-repeat scroll bottom;
    color: white;
    border-radius: 15px;
    box-shadow: 0 0 15px black;
}
.time-article h1{
    font-size: 30px;
}
```

JavaScript File

```
// Display a Digital Clock on the web page
function indianClock() {
    var today = new Date();
    var options = {timeZone : 'Asia/Kolkata'};
    var time = today.toLocaleTimeString('en-US',options);
    var date = today.toLocaleDateString('en-US',options);
    document.querySelector('#india-time').textContent = time;
    document.querySelector('#india-date').textContent = date;
}
setInterval(indianClock,1000);

// USA Clock
function usaClock() {
    var today = new Date();
    var options = {timeZone : 'America/New_York'};
    var time = today.toLocaleTimeString('en-US',options);
    var date = today.toLocaleDateString('en-US',options);
    document.querySelector('#us-time').textContent = time;
    document.querySelector('#us-date').textContent = date;
}
setInterval(usaClock,1000);
// China Clock
function chinaClock() {
    var today = new Date();
    var options = {timeZone : 'Asia/Shanghai'};
    var time = today.toLocaleTimeString('en-US',options);
    var date = today.toLocaleDateString('en-US',options);
    document.querySelector('#china-time').textContent = time;
    document.querySelector('#china-date').textContent = date;
}
setInterval(chinaClock,1000);
```

Result:

Math Objects

Math Object	Usage	Result	Description
Math.round()	Math.round(4.7) Math.roudn(4.4)	5 4	Math.round(x) returns the value of x rounded to its nearest integer
Math.pow()	Math.pow(8,2)	64	Math.pow(x, y) returns the value of x to the power of y
Math.sqrt()	Math.sqrt(64)	8	Math.sqrt(x) returns the square root of x.
Math.abs()	Math.abs(-4.4)	4	Math.abs(x) returns the absolute (positive) value of x
Math.ceil()	Math.ceil(4.4)	5	Math.ceil(x) returns the value of x rounded **up** to its nearest integer
Math.floor()	Math.floor(4.7)	4	Math.floor(x) returns the value of x rounded **down** to its nearest integer.
Math.min()	Math.min(0,20,-10,200)	-10	Math.min() can be used to find the lowest value in a list of arguments
Math.max()	Math.max(200,-10,300)	300	Math.max() can be used to find the highest value in a list of arguments
Math.random()	Math.random()	0.234234	Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)
Math.PI	Math.PI	3.14159	Constant value.

JavaScript Regular Expressions

- A regular expression is a sequence of characters that forms a **search pattern**. When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of **text search** and **text replaces** operations.

RegularExpression	Description
[abcdef]	abcdef allowed here rest of the characters are not allowed here.
[a-d]	abcd allowed here rest of the characters are not allowed here.
[0-9]	0 to 9 digits are allowed here.
{3,15}	Length at least 3 characters and maximum length of 15
[aA-zZ]	the starting character is in the lowercase or uppercase alphabet.
[^0-9]	matches any non-digit
[+-]?	matches an optional "+", "-", or an empty string.
{m}	{m}: exactly m times
{m,}	m or more (m+)

Ex:

```
// UserName RegEx Validation
if(!userNameElement.value.match(/^[A-Za-z0-9_]{1,15}$/)){
    message = 'RegExp is Not Matched';
}
```

- The above example the username will accept small (a to z), capital (A to Z), digits (0 to 9) & underscore will accept the username must contains in between 1 to 15 characters only.

Functions

- A function can be defined as a block of code which is used to perform certain task and which takes zero parameters or few parameters and performs some processing and returns a processed value.

JavaScript Function Syntax

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

Syn:

```
function function_name(arg1,arg2…){
        ------------------------
        ------------------------
 }
```

Ex: the following code defines a simple function named square:

```
<script>
    function square(number){
        return number*number;
    }
    var square_temp=square(10);
    console.log("10 Square is "+square_temp);
</script>
```

Result: 10 Square is 100

- Here, in the above example square is the function name which takes one parameter is number and the function consists of return statement which returns some value by the function.
- Primitive parameters (such as a number) are passed to functions **by value**; the value is passed to the function, but if the function changes the value of the parameter, **this change is not reflected globally or in the calling function**.

```
<script>
    function myFunc(theObject) {
        theObject.make   = 'Toyota';
        console.log(theObject.make);
    }

    var mycar = {make: 'Honda', model: 'Accord', year: 1998};
    var x, y;

    x = mycar.make; // x gets the value "Honda"
    console.log(x);

    myFunc(mycar);

    y = mycar.make; // y gets the value "Toyota"
    console.log(y); // (the make property was changed by the function,
```

- If you pass an object (i.e. a non-primitive value, such as Array or a user-defined object) as a parameter to the function and the function changes the object's properties, that change is visible outside the function, as shown in the above example.

Function Expression or anonymous Function:

- In JavaScript we are not mention the function name for creating the function definition and this function definition pass as a value to the variable is called as ***function expression or anonymous function***. The below example defines function expression or anonymous function.

```
<!--Function expression or anonymous function-->
<script>
    var fun=function() {
        var ename='Satish';
        var salary=90000;
        console.log("Employee Name is "+ename+" and his salary is "+salary);
    };
    console.log(fun('satish'));
</script>
```

Result:

Employee Name is Satish and his salary is 90000

- However, a name can be provided with a function expression and can be used inside the function to refer to itself, or in a debugger to identify the function in stack traces:

```
<script>
    var factorial = function fac(n) {
        return n < 2 ? 1 : n * fac(n - 1);
    };
    console.log(factorial(6));
</script>
```

Result: 720

Passing arguments in functions

- In JavaScript programming, a function is defined with some parameters; we can even pass less no. of parameters and also we can pass more no. of parameters to the same function both are accepted.
- JavaScript will accept if you pass the less no. of parameters then the remaining parameters will assign with *undefined value*.

```
<!--pass less no. of parameters to the function-->
<script>
    function employee(name,location,salary) {
        return 'Employee name is '+name+' and his working ' +
            'location is '+location+' and his salary is '+salary;
    }
    var employeedata=employee('Satish','Hyderabad');
    console.log(employeedata);
</script>
```

Result:

*Employee name is **Satish** and his working location is **Hyderabad** and his salary is **undefined***

- JavaScript will accept if you pass more no. of parameters then the extra parameters will be ignored by the function.

```
<!--pass more no. of parameters to the function-->
<script>
    function employee(name,location,salary) {
        return 'Employee name is '+name+' and his working ' +
            'location is '+location+' and his salary is '+salary;
    }
    var employeedata=employee('Satish','Hyderabad',80000,'Associate Engineer');
    console.log(employeedata);
</script>
```

Result:

*Employee name is **Satish** and his working location is **Hyderabad** and his salary is **80000***

this Keyword

- **this** can be defined as to refers the current object or current class variables or functions.
- In JavaScript we can even write a function inside an object, within that function we can access the properties by using the keyword called **this**.
- Usually for every function if you pass more no. of parameters then the extra parameters are just ignored by the function but in JavaScript for each function definition for an default object available called arguments, this object holds all the parameters that we pass for that function even non defined those parameters in the function declaration.

```
// Functions in Objects and Execution
var student = {
    firstName : 'Arjun',
    lastName : 'Reddy',
    fullName : function() {
        return this.firstName + " " + this.lastName;
    }
};
console.log(student.fullName());
```

this Alone

- When used alone, **this** refers to the Global object.
- In a browser window the Global object is [**object Window**]:

```
var x=this;
console.log(x);       //Window Object

function this_key(){
  return this;
}
console.log(this_key()); //Window Object
```

Use strict Mode

- When used in a function, **this** also refers to the Global object. The Global object is the default binding:
- In **strict mode**, when used in a function, **this** is **undefined**. Strict mode does not allow default binding:

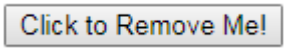
```
"use strict";
var x=this;
console.log(x);       //Window Object

function this_key(){
  return this;
}
console.log(this_key()); //undefined
```

this in Event Handlers

- In HTML event handlers, this refers to the HTML element that received the event:

```
<button onclick="this.style.display='none'">
     Click to Remove Me!
</button>
```

Before click the button:



After click the button:

Nothing shown

Callback function

- **Function** can be passed as an argument to other **functions**, can be returned by another **function** and can be assigned as a value to a variable.

```
function sayHello() {
   return "Hello, ";
}
function greeting(helloMessage, name) {
  console.log(helloMessage() + name);
}
// Pass `sayHello` as an argument to `greeting` functior
greeting(sayHello, "JavaScript!");
```

- We are passing our sayHello() function as an argument to the greeting()function, this explains how we are treating the function as a value. The function that we pass as an argument to another function, called a Callback function**.** *sayHello is a Callback function.*

Example_1:

```
var employee={
    firstName:'Sudheer',
    lastName:'Andra',
    address:{
        city:'Hyd',
        state:'Telangana',
        country:'India'
    },
    isFromCity:function (city) {
        return city.toLowerCase()===this.address.city.toLowerCase();
    },
    isFromState:function (state) {
        return state.toLowerCase()===this.address.state.toLowerCase();
    }
};
console.log('is From HYD: '+employee.isFromCity('HYD'));
console.log('is From TelaNGana: '+employee.isFromState('TelaNGana'));
```

Result:

```
is From HYD: true
is From TelaNGana: true
```

Default Parameters

- **Default function parameters** allow named parameters to be initialized with default values, if no value is passed to the function or passed *undefined*. Then default value will take as function argument.

```
function multiply(a,b=1){
  return a*b;
}
console.log(multiply(5,2));    //10
console.log(multiply(5));      //5
```

Here in the above example multiply function *b* parameter with the default value is **1** if b value is not passed to the function.
Example:

```
function test(num=1){
  console.log(typeof num);
}
test();           //output: number
test(undefined);  //output: number
test('');         //output: string
test(null);       //output: object
```

Return Statement:

- The return statement stops the execution of a function and returns a value from that function.

```
function getRectArea(width,height){
  if(width>0 && height>0){
    return width*height;
  }
  return 0;
}
console.log(getRectArea(3,4));
console.log(getRectArea(-3,4));
```

- When a return statement is used in a function body, the execution of the function is stopped at return statement. If specified, a given value is returned to the function caller. For example, the following function returns the square of its argument, x, where x is a number.

```
function square(x) {
   return x * x;
}
var demo = square(3);
console.log(demo);
// demo will equal 9
```

- If the value is omitted, undefined is returned instead.

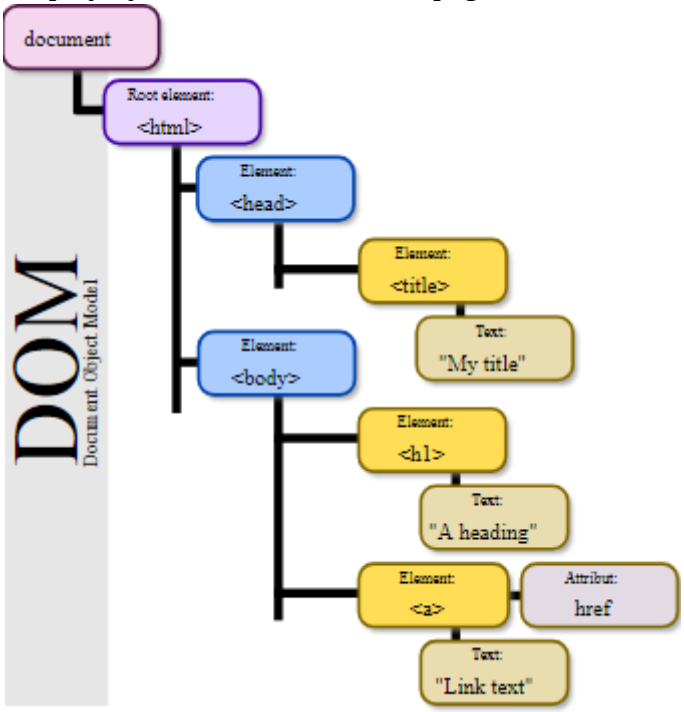
```
function square(x){
  return;
}
console.log(mul(5));
//output: undefined
```

JavaScript DOM Manipulation:

While we execute the HTML Document on a browser, the browser does the following things:

- ✓ The browser scans the entire HTML Document.
- ✓ The browser prepares the DOM structure with HTML Elements.
- ✓ Depending on the structure browser displays the content on the webpage.

- Each HTML Document represents a DOM Structure. In the DOM all the HTML elements are Objects. Based on those objects the browser will construct the DOM tree.
- Each Browser contains a collection of HTML Web pages, Each Webpage is having a DOM Structure, and Collection of all DOM Structure is named as BOM (Browser Object Model). The Top level Object in the Browser Object Model is window.
- By using JavaScript window object we can perform the browser related operations, by using Document Object we can manipulate the entire HTML Document with JavaScript Processing Logic so that we can display dynamic content on web page.



- In DOM Manipulation process we can access the individual HTML Element and apply the processing logic and display the processed value back to HTML.
- JavaScript allows you to create a dynamic update content and it can change all the HTML Element, attributes and CSS Styles in the page.
- JavaScript can remove the existing HTML element and attributes. JavaScript can add the HTML Elements and attribute. JavaScript can create new HTML Events in the page

Usage	JQuery	DOM Operations
Good Morning	doc.querySelector('h2');	$('h2')
<h2 id='test'>Morning</h2>	doc.querySelector('#test');	$('#test')
<h2 class='abc'>Good Morning</h2>	doc.querySelector('.abc');	$('.abc')
Good Morning	var h1Eelement=doc.querySelector('h1');	$('h1').text();

DOM Elements

- getElementsByTagName
- getElementById
- getElementByClass

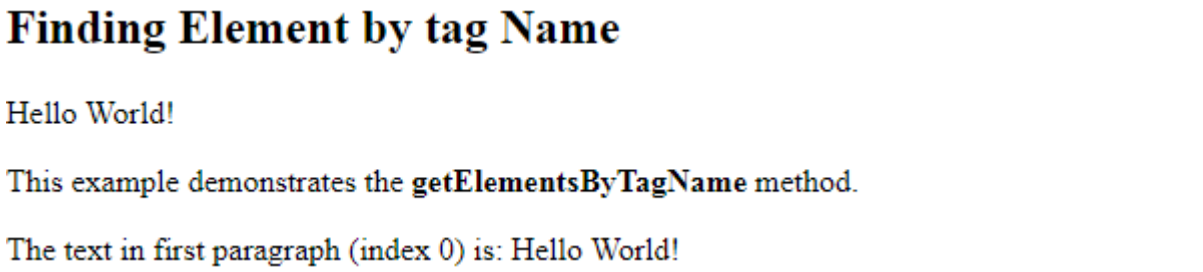
HTML Elements by Tag Name

- **getElementsByTagName** is used to select related tags in the HTML Document.
- Using **getElementsByTagName** we can only select the *tag Names.*

Example:

```
<h2>Finding Element by tag Name</h2>
<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
<script>
    var x=document.getElementsByTagName("p");
    document.getElementById("demo").innerHTML='The text in first paragraph (index 0) is:
    +x[0].innerHTML;
</script>
```

Result:



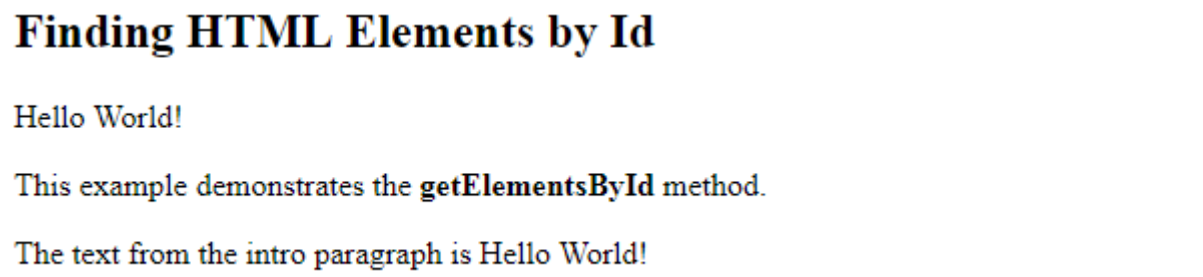
HTML Element by Id:

- getElementsById is used to find ***id*** in the HTML Document. Using getElementsById we can only select the *id.*
- This example finds the element with id="intro":

Example:

```
<h2>Finding HTML Elements by Id</h2>
<p id="intro">Hello World!</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>

<p id="demo"></p>

<script>
    var myElement = document.getElementById("intro");
    document.getElementById("demo").innerHTML =
    "The text from the intro paragraph is " + myElement.innerHTML;
</script>
```

Result:



- If the element is found, the method will return the element as an object (in myElement).
- If the element is not found, myElement will contain null

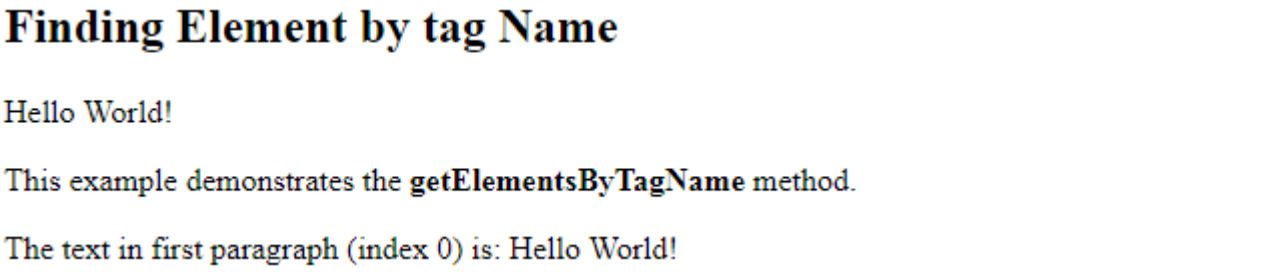
HTML Elements by Class Name

- getElementsByClassName is used to find all the same class names in the HTML Document.
- Using getElementsByClassName we can only select class names.

Example:

```
<h2>Finding Element by tag Name</h2>
<p class="test">Hello World!</p>
<p class="test">This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
<script>
    var x=document.getElementsByClassName("test");
    document.getElementById("demo").innerHTML='The text in first paragraph (index 0) is: '
        +x[0].innerHTML;
</script>
```

Result:



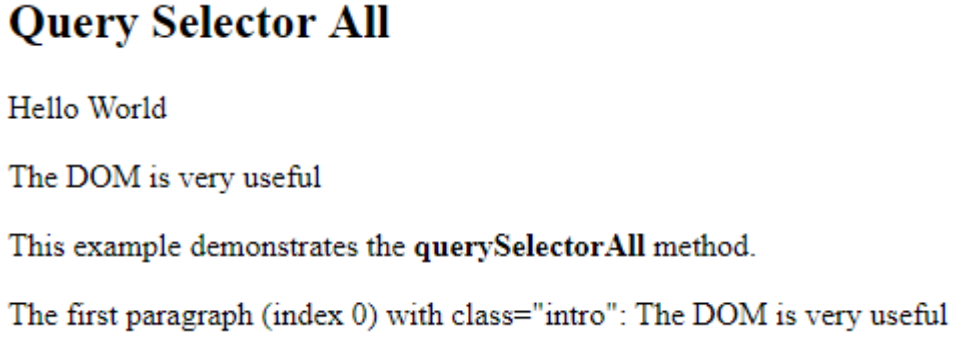
HTML Elements by CSS Selectors

- If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

Example:

```
<h2>Query Selector All</h2>
<p>Hello World</p>
<p class="test">The DOM is very useful</p>
<p class="test">This example demonstrates the
    <b>querySelectorAll</b> method.</p>
<p id="demo"></p>

<script>
    var x=document.querySelectorAll('p.test');
    document.getElementById('demo').innerHTML='The first ' +
        'paragraph (index 0) with class="intro": '+x[0].innerHTML;
</script>
```

Result:



Example:

```
<form id="frm1" action="/action_page.php">
    First name: <input type="text" name="fname" value="Donald"><br>
    Last name: <input type="text" name="lname" value="Duck"><br><br>
    <input type="submit" value="Submit">
</form>
<p>Click "Try it" to display the value of each element in the form.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
    function myFunction() {
        var x=document.forms['frm1'];
        var temp='';
        for(var i=0;i<x.length;i++){
            temp+=x.elements[i].value+"<br>";
        }
        document.getElementById('demo').innerHTML=temp;
    }
</script>
```

Result:

First name: Donald
Last name: Duck

Submit

Click "Try it" to display the value of each element in the form.

Try it

Donald
Duck
Submit

InnerHTML

- It is used to change the content of the HTML Element as well as change the attribute value of the HTML Element by using *innerHTML*
- The below Example demonstrate Change the content of the p tag let us see..

HTML*:*

```
<body>

<h2>JS DOM Manipulation</h2>

<script src="10_script.js"></script>

</body>
```

JS*:*

```
//Change the Content of h2 Element
    document.querySelector('h2').innerHTML = 'DOM Manipulation';
```

Result:

DOM Manipulation

DOM EventListener

The HTML DOM allows you to execute code when an event occurs. A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element & mouse over the HTML Element etc…

Events are generated by the browser when "things happen" to HTML elements:

- When a user clicks the mouse
- When a web page has loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Resource events

Event Name	Description
cached	The resources listed in the manifest have been downloaded, and the application is now cached.
error	A resource failed to load.
abort	The loading of a resource has been aborted.
load	A resource and its dependent resources have finished loading.
beforeunload	The window, the document and its resources are about to be unloaded.
unload	The document or a dependent resource is being unloaded.

Network events

Event Name	Description
online	The browser has gained access to the network.
offline	The browser has lost access to the network.

Focus events

Event Name	Description
focus	An element has received focus (does not bubble).
blur	An element has lost focus (does not bubble).

WebSocket events

Event Name	Description
open	A WebSocket connection has been established.
message	A message is received through a WebSocket.
error	A WebSocket connection has been closed with prejudice (some data couldn't be sent for example).
close	A WebSocket connection has been closed.

Session History events

Event Name	Description
pagehide	A session history entry is being traversed from.
pageshow	A session history entry is being traversed to.
popstate	A session history entry is being navigated to (in certain cases).

CSS Animation events

Event Name	Description
animationstart	A CSS animation has started.
animationend	A CSS animation has completed.
animationiteration	A CSS animation is repeated.

CSS Transition events

Event Name	Description
transitionstart	A CSS transition has actually started (fired after any delay).
transitioncancel	A CSS transition has been cancelled.
transitionend	A CSS transition has completed.
transitionrun	A CSS transition has begun running (fired before any delay starts).

Form events

Event Name	Description
reset	The reset button is pressed
submit	The submit button is pressed

Printing events

Event Name	Description
beforeprint	The print dialog is opened
afterprint	The print dialog is closed

Text Composition events

Event Name	Description
compositionstart	The composition of a passage of text is prepared (similar to keydown for a keyboard input, but works with other inputs such as speech recognition).
compositionupdate	A character is added to a passage of text being composed.
compositionend	The composition of a passage of text has been completed or canceled.

View events

Event Name	Description
fullscreenchange	An element was turned to fullscreen mode or back to normal mode.
fullscreenerror	It was impossible to switch to fullscreen mode for technical reasons or because the permission was denied.
resize	The document view has been resized.
scroll	The document view or an element has been scrolled.

Clipboard events

Event Name	Description
cut	The selection has been cut and copied to the clipboard
copy	The selection has been copied to the clipboard
paste	The item from the clipboard has been pasted

Keyboard events

Event Name	Description
keydown	ANY key is pressed
keypress	ANY key except Shift, Fn, CapsLock is in pressed position. (Fired continously.)
keyup	ANY key is released

Mouse events

Event Name	Description
auxclick	A pointing device button (ANY non-primary button) has been pressed and released on an element.
click	A pointing device button (ANY button; soon to be primary button only) has been pressed and released on an element.
contextmenu	The right button of the mouse is clicked (before the context menu is displayed).
dblclick	A pointing device button is clicked twice on an element.
mousedown	A pointing device button is pressed on an element.
mouseenter	A pointing device is moved onto the element that has the listener attached.
mouseleave	A pointing device is moved off the element that has the listener attached.
mousemove	A pointing device is moved over an element. (Fired continously as the mouse moves.)
mouseover	A pointing device is moved onto the element that has the listener attached or onto one of its children.
mouseout	A pointing device is moved off the element that has the listener attached or off one of its children.
mouseup	A pointing device button is released over an element.
pointerlockchange	The pointer was locked or released.
pointerlockerror	It was impossible to lock the pointer for technical reasons or because the permission was denied.
select	Some text is being selected.
wheel	A wheel button of a pointing device is rotated in any direction.

Progress events

Event Name	Description
abort	Progression has been terminated (not due to an error).
error	Progression has failed.
load	Progression has been successful.
loadend	Progress has stopped (after "error", "abort" or "load" have been dispatched).
loadstart	Progress has begun.
progress	In progress.
timeout	Progression is terminated due to preset time expiring.

Drag & Drop events

Event Name	Description
drag	An element or text selection is being dragged (Fired continuously every 350ms).
dragend	A drag operation is being ended (by releasing a mouse button or hitting the escape key).
dragenter	A dragged element or text selection enters a valid drop target.
dragstart	The user starts dragging an element or text selection.
dragleave	A dragged element or text selection leaves a valid drop target.
dragover	An element or text selection is being dragged over a valid drop target. (Fired continuously every 350ms.)
drop	An element is dropped on a valid drop target.

Media events

Event Name	Description
audioprocess	The input buffer of a ScriptProcessorNode is ready to be processed.
canplay	The browser can play the media, but estimates that not enough data has been loaded to play the media up to its end without having to stop for further buffering of content.
canplaythrough	The browser estimates it can play the media up to its end without stopping for content buffering.
complete	The rendering of an OfflineAudioContext is terminated.
durationchange	The duration attribute has been updated.
emptied	The media has become empty; for example, this event is sent if the media has already been loaded (or partially loaded), and the load() method is called to reload it.
ended	Playback has stopped because the end of the media was reached.
loadeddata	The first frame of the media has finished loading.
loadedmetadata	The metadata has been loaded.
pause	Playback has been paused.
play	Playback has begun.
playing	Playback is ready to start after having been paused or delayed due to lack of data.
ratechange	The playback rate has changed.
seeked	A *seek* operation completed.
seeking	A *seek* operation began.
stalled	The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
suspend	Media data loading has been suspended.
timeupdate	The time indicated by the currentTime attribute has been updated.
volumechange	The volume has changed.
waiting	Playback has stopped because of a temporary lack of data.

Scopes in JavaScript

- A Scope in JavaScript indicates the accessibility area of any variable. In Programming you create a scope for any variables in two ways:
    - Block Scoping
    - Function Scoping

Block Scoping

- We create a scope for any variable using block scoping is we can create the variables inside a statement block or if condition or for loop etc...

Statement Block:

```
{
    var a=10;
}
```

if-condition

```
if(100===100){
    var user='guest';
}
```

- In the above cases the variable which is declare inside a block or if condition which can't be accessed outside the, if condition or block.
- We can even create scopes in programming using for loop or while loop or do-while loop.

  Ex:

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

- The variable i will only be accessed inside for loop because the scope of *i* variable will be up to for loop only.

Function Scoping:

- We can even create scopes using functions as follows:

  Ex:

```
function greet(){
    var greetMsg="Good Morning";
}
greet();
```

- In the above example the variable greet message is only accessible inside a function is called function scoping.

Note

- In JavaScript Scoping there are two kinds of scopes like parent scope and child scope.
- If any variable which is declare outside the block or function is parent scoped variable ***(global)*** and any variable which is declared within the block or function is child scope ***(local)*** variable.

  Ex:

```
var b=10;                          //parent scope
function greet(){
    var greetMsg="Good Morning"; //child scope
}
greet();
```

- Any variable which are declared in the parent scope are also accessible in the child scope but child scope data which can't be accessed outside the defined scope.
- Any variable which is declared outside the function is also called global scoped variable and in JavaScript the top most object is window object.

Window object

- Any variable declared in the global scope will be added as a property for the window object.
- JavaScript not supports block scoping even thought we declared the variable inside a statement block or if condition or for loop those will be treated as the global variables, in JavaScript all the Global variables will be created as a property for the window object. It means even though we create the variables if block or for block but still they will be created as a properties of window object.
- Creating variables in the window object is a bad idea because any variable declared with window object that can be accessible for all the JavaScript Files in the entire website.
- To avoid creating the variables in the window object we can create a separate scope for those variables by using only way to declare the variables inside a function.
- If any variable which is commonly used for all the functions of the JavaScript program we can create that variable in the global scope and if any variable we don't want to access for any other function we can create that variable inside the required function.

IIFE (Immediately Invoked Function Expression)

Ex:

```
var b=10;                          //parent scope
function greet(){
    var greetMsg="Good Morning"; //child scope
}
greet();
```

- In the above example a variable as been declared in the global scope and the greet message variable is declared in the greet function scope in this case the variable will be added in the window object and the greet message variable will be hidden from the window object.
- But in the above program each JavaScript function itself is a variable that holds the value called function.
- So in the above example the greet function as been declared globally and not declared inside any function in this case the greet variable will be created as a property in the window object, if it is there in the window object from any JavaScript file we can able to call the function.
- In order to avoid creating that variable in the window object we can execute that function immediately as soon as it is declared without having any name is the concept called IIFE. In this approach the function name is not visible in the window object.

Ex:

```
(function(){
  var greetMsg="Good Morning";
  console.log(greetMsg);
})();
```

Read & Write Operations

- If we declare any variable with some value is called as write operation.

Ex:

```
var a=10;       //Write operation
```

- If you read the value of a variable and printing to console or passed some other function is called read operation

Ex:

```
console.log(a);     //read operation
```

- In any programming we can perform the only read and write operations with the variables.
- For any programming language apart from JavaScript like C,C++ & Java any operation we are going to perform with the variable the first rule is first we have to declare the variable and use the variable.
- But in JavaScript the variable declaration is purely depend on what operation we are going to perform with the variable.
- In JavaScript the write operation is possible for non declared variables but the read operation is not possible for the non declared variables.

```
var a=10;         //Write operation possible
console.log(b);   //RE: 'b' not defined
```

Compilation and Interpretation in JavaScript

- The Programming languages like C, C++ & Java are known as the compile and interpreted programming languages. It means a Java program don't execute automatically, first we have to compile the Java program manually and execute the Java program manually.
- But in JavaScript the JavaScript program doesn't require any intermediary file, the JavaScript program executes as it is in the Run time Environment called browser.
- In JavaScript also compilation and interpretation or execution happens in the same file.
- While we execute the JavaScript program on the browser first the compilation step happens followed by interpretation step in the same JavaScript file.

- The ***Compilation process*** the JavaScript performs the following things:
    - Variable Declaration
    - function Declaration
    - Scope chain creation using the variable and function names

- In the ***interpretation*** step the following things will happens:
    - Usage of scope chain created in the compilation step
    - The variable assignments
    - The actual function executions.

- While we execute a JavaScript file on the browser the browser scans the JavaScript program two times, the first for the compilation step and next for the interpretation step.

ES6 Features

From the ES6 Version ECMA Company released new language features as follows:

1. Default Parameters
2. Template Strings
3. Let and const keywords
4. for-of loop
5. Lambda Expre's
6. Spread operator
7. Destructuring

Note: All the above features are explained in typescript tutorial. Please refer typescript tutorial.

Interview Questions & Answers

1. What is JavaScript?

→ JavaScript is a client-side as well as server side scripting language that can be inserted into HTML pages and is understood by web browsers.

2. What is the differences between Java and JavaScript?

→ Java is a complete programming language. JavaScript is a coded program that can be introduced to HTML pages.

→ Java is an object - oriented programming (OOPS) or structured programming language like C++ or C whereas JavaScript is a client-side scripting language.

→ Java creates applications that run in a virtual machine or browser while JavaScript code is run on a browser only.

→ Java code needs to be compiled while JavaScript code are all in text.

→ Java is strongly typed language whereas JavaScript is a weekly typed language.

→ Java program uses more memory while JavaScript requires less memory therefore it is used in web pages.

→ Java has a thread based approach to concurrency but JavaScript has event based approach to concurrency.

3. What are JavaScript Data Types?

Following are the JavaScript Data types:

- Number
- String
- Boolean
- Object
- Undefined

4. What is the use of isNaN function?

→ isNan function returns true if the argument is not a number otherwise it is false.

5. Between JavaScript and an ASP script, which is faster?

→ JavaScript is faster. JavaScript is a client-side language and thus it does not need the assistance of the web server to execute. On the other hand, ASP (Active Server Pages) is a server-side language and hence is always slower than JavaScript. JavaScript now is also a server side language (nodejs).

6. What is negative infinity?

→ Negative Infinity is a number in JavaScript which can be derived by dividing negative number by zero.

7. What are undeclared and undefined variables?

→ Undefined variables are those that are declared in the program but have not been given any value. If the program tries to read the value of an undefined variable, an undefined value is returned.

→ Undeclared variables are those that do not exist in a program and are not declared. If the program tries to read the value of an undeclared variable, then a runtime error is encountered.

8. Write the code for adding new elements dynamically?

Ex:

```
<html>
<head>
    <title>t1</title>
    <script type="text/javascript">
        function addNode() {
            var newP = document.createElement("h1");
            var textNode = document.createTextNode(" This is a new text node");
            newP.appendChild(textNode);
            document.getElementById("demo").appendChild(newP);
        }
    </script>
</head>
<body>
    <p id="demo" onclick="addNode()">firstP<p>
</body>
</html>
```

Result:

firstP

This is a new text node

Note: When you click on firstp the below text is shown.

9. What are global variables? How are these variable declared and what are the problems associated with using them?

→ A variable that is declared outside of a function definition is called a global variable and its scope is throughout your program means its value is accessible and modifiable throughout your program.

→ The most common problem is updating the global variable. If more than one function is trying to update the variable, then we have to be cautious that it is done in the desired manner.

Ex:

```
<script>
    // Global variable.
    var a = "Dot Net Tricks !";
    function show() {
        // A Local variable is declared in this function.
        var a = "Hello World !";
        alert("Value of 'a' inside the function :" + a); //Hello World !

        //b will have global scope
        b = "Hello JavaScript !";
        Display();
    }
    function Display() {
        //Since b variable has global scope
        alert("Value of 'b' outside the function : " + b); //Hello JavaScript !
    }
    alert("Value of 'a' outside the function : " + a); //Dot Net Tricks !
    show(); // Value of 'a' inside the function : Hello World !
            // Value of 'b' outside the function : Hello JavaScript !
</script>
```

→ The problems that are faced by using global variables are the clash of variable names of local and global scope. Also, it is difficult to debug and test the code that relies on global variables.

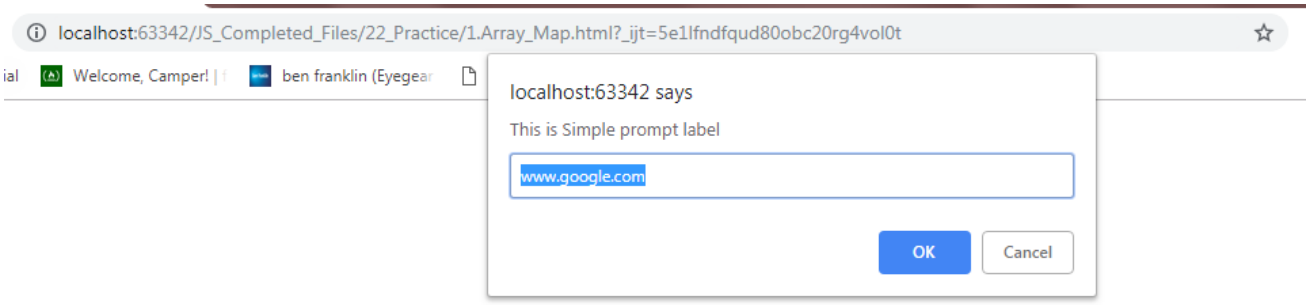
10. What is a prompt box?

A prompt box is a box which allows the user to enter input by providing a text box. Label and box will be provided to enter the text or number.

Ex:

```
<script>
    prompt("This is Simple prompt label","www.google.com");
</script>
```

Result:



11. What is 'this' keyword in JavaScript?

→ 'this' keyword refers to the object from where it was called.

→ "this" refers to new instance

→ When a function is invoked with "new" keyword then the function is known as constructor function and returns a new instance. In such cases, the value of "this" refers to newly created instance.

Ex:

```
function Person(fn, ln) {
        this.first_name = fn;
        this.last_name = ln;

        this.displayName = function() {
                console.log(`Name: ${this.first_name} ${this.last_name}`);
        }
}

let person = new Person("John", "Reed");
person.displayName();  // Prints Name: John Reed
let person2 = new Person("Paul", "Adams");
person2.displayName();  // Prints Name: Paul Adams
```

12. Explain the working of timers in JavaScript? Also elucidate the drawbacks of using the timer, if any?

→ Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functions setTimeout, setInterval and clearInterval.

- The setTimeout(function, delay) function is used to start a timer that calls a particular function after the mentioned delay.

Ex:

```
<script>
    var message = function(){
      console.log("greeting from setTimeOut")
    };
    setTimeout(message,5000);
</script>
```

After 5 seconds message will be displayed on the console.

- The setInterval(function, delay) function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled.
- The clearInterval(id) function instructs the timer to stop.

→ Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

13. Which symbol is used for comments in JavaScript?

→ All programs should be commented in such a manner as to easily describe the purpose of the code. Comments are specially marked lines of text in the program that are **not executed**.

→ Using comments that is easily understand by the other programmers also and it will improve the readability.

```
// for Single line comments and

/* Multi
Line
Comment*/
```

14. What is the difference between ViewState and SessionState?

→ '*ViewState*' is specific to a page in a session. When another page is loaded, the previous page data is no longer available.

→ '*SessionState*' is specific to user, specific data that can be accessed across all pages in the web application. SessionState is the data of a user session and is maintained on the server side. This data available until user closes the browser or session time-outs.

15. Explain how can you submit a form using JavaScript?

→ In javascript onclick event is used to submit the form, you can use form.submit() method to submit form.

→ You can perform submit action by, submit button, by clicking on hyperlink, button and image tag etc. You can also perform javascript form submission by form attributes like id, name, class, tag name as well.

Html File:

```
<form action="#" method="post" name="form_name" id="form_id" class="form_class" >
    <h2>Javascript Form Submit Example</h2>
    <label>Name :</label>
    <input type="text" name="name" id="name" placeholder="Name" />
    <label>Email :</label>
    <input type="text" name="email" id="email" placeholder="Valid Email" />
    <input type="button" name="submit_id" id="btn_id" value="Submit by Id"
           onclick="submit_by_id()"/>
    <input type="button" name="submit_name" id="btn_name" value="Submit by Name"
           onclick="submit_by_name()"/>
    <input type="button" name="submit_class" id="btn_class" value="Submit by Class"
           onclick="submit_by_class()"/>
    <input type="button" name="submit_tag" id="btn_tag" value="Submit by Tag"
           onclick="submit_by_tag()"/>
</form>
```

Result:

16. Does JavaScript support automatic type conversion?

→ Yes JavaScript does support automatic type conversion, it is the common way of type conversion used by JavaScript developers.

17. How can the style/class of an element be changed?

→ Using following statement the style elements can be changed.

```
Syn:
    document.querySelector('id/class/tagNmae').style.properyname="properyvalue";
Ex:
    document.querySelector('#demo').style.fontSize="20px";

                    Or

    document.getElementById('myElement').className = "myclass";
```

18. Explain how to read and write a file using JavaScript?

There are two ways to read and write a file using JavaScript

- Using JavaScript extensions (runs from JavaScript Editor)
- Using a web page and Active X objects (Internet Explorer only)

In JavaScript Extensions, you can use

```
var file = fopen(getScriptPath(), 0); to open a file
```

The function fread() is used for reading the file content. The function fwrite() is used to write the contents to the file.

```
str = fread(file,flength(file) ;
```

Using ActiveX objects, following should be included in your code to read a file:

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var s = fso.OpenTextFile("C:\\example.txt", 1, true);
```

19. What are all the looping structures in JavaScript?

Following are looping structures in JavaScript:

→ For
→ While
→ do-while loops

20. What is called Variable typing in JavaScript?

→ Variable typing is used to assign a number to a variable and the same variable can be assigned to a string.

Example

```
i = 10;
i = "string";
```

This is called variable typing.

21. How can you convert the string of any base to integer in JavaScript?

→ The parseInt() function is used to convert numbers between different bases. parseInt() takes the string to be converted as its first parameter, and the second parameter is the base of the given string.

→ In order to convert 4F (of base 16) to integer, the code used will be -parseInt ("4F", 16);

22. Explain the difference between "==" and "==="?

→ "==" checks only for equality in value whereas "===" is a stricter equality test and returns false if either the value or the type of the two variables are different.

23. What would be the result of 3+2+"7"?

→ Since 3 and 2 are integers, they will be added numerically. And since 7 is a string, its concatenation will be done. So the result would be 57.

24. Explain how to detect the operating system on the client machine?

→ In order to detect the operating system on the client machine, the navigator.platform string (property) should be used.

25. What do mean by NULL in Javascript?

→ The NULL value is used to represent no value or no object. It implies no object or null string, no valid boolean value, no number and no array object.

26. What is the function of delete operator?

→ The delete keyword is used to delete the property as well as its value.

Ex:

```
var student= {age:20, batch:"ABC"};
delete student.age;
```

27. What are all the types of Pop up boxes available in JavaScript?

→ Alert
→ Confirm and
→ Prompt

28. What is the use of Void(0)?

→ Void(0) is used to prevent the page from refreshing and parameter "zero" is passed while calling.
→ Void(0) is used to call another method without refreshing the page.

29. How can a page be forced to load another page in JavaScript?

Yes. In the javascript code:

```
window.location.href = "http://new.website.com/that/you/want_to_go_to.html";
```

30. What is the data type of variables of in JavaScript?

→ All variables in the JavaScript are object data types.

31. What is the difference between an alert box and a confirmation box?

→ An alert box displays only one button which is the OK button.
→ But a Confirmation box displays two buttons namely OK and cancel.

32. What are escape characters?

→ Escape characters (Backslash) is used when working with special characters like single quotes, double quotes, apostrophes and ampersands. Place backslash before the characters to make it display.

Ex:

```
document.write("I m a "good" boy")
document.write("I m a \"good\" boy")
```

33. What are JavaScript Cookies?

→ Cookies are the small test files stored in a computer and it gets created when the user visits the websites to store information that they need. Example could be User Name details and shopping cart information from the previous visits.

34. Whether JavaScript has concept level scope?

→ No. JavaScript does not have concept level scope. The variable declared inside the function has scope inside the function.

35. Mention what is the disadvantage of using innerHTML in JavaScript?

→ If you use innerHTML in JavaScript the disadvantage is Content is replaced everywhere
→ We cannot use like "appending to innerHTML"
→ Even if you use +=like "innerHTML = innerHTML + 'html'" still the old content is replaced by html
→ The entire innerHTML content is re-parsed and build into elements, therefore its much slower The innerHTML does not provide validation and therefore we can potentially insert valid and broken HTML in the document and break it.

36. What is break and continue statements?

→ Break statement exits from the current loop. Whereas Continue statement continues with next statement of the loop.

37. What are the two basic groups of dataypes in JavaScript?

→ They are as – Primitive & Reference types.
→ Primitive types are number and Boolean data types. Reference types are more complex types like strings and dates.

38. How generic objects can be created?

→ Generic objects can be created as:

```
var name = new object();
```

39. What is the use of type of operator?

→ 'typeof' is an operator which is used to return a string description of the type of a variable.

40. Which keywords are used to handle exceptions?

→ Try… Catch---finally is used to handle exceptions in the JavaScript.
→ try block is used to identify the errors and generate one exception object and send it to the catch block then immediately catch will identify that exception and handle it. Finally block is executed whether try/catch block may or may not execute.

```
try{
    //Code
}
catch(exp){
    //Code to throw an exception
}
finally{
    //Code runs either it finishes successfully or after catch
}
```

41. What is the use of blur function?

→ Blur function is used to remove the focus from the specified object.

42. What are the different types of errors in JavaScript?

There are three types of errors:

→ ***Load time errors:*** Errors which come up when loading a web page like improper Syntax errors are known as Load time errors and it generates the errors dynamically.
→ ***Run time errors:*** Errors that come due to misuse of the command inside the HTML language.
→ ***Logical Errors:*** These are the errors that occur due to the bad logic performed on a function which is having different operation.

43. What is the difference between JavaScript and Jscript?

Both are almost similar. JavaScript is developed by Netscape and Jscript was developed by Microsoft

44. What is the way to get the status of a CheckBox?

→ alert is used to get the status of a checkbox by the following way:

```
alert(document.getElementById('checkbox1').checked);
```

→ If the CheckBox will be checked, this alert will return TRUE.

45. Explain window.onload and onDocumentReady?

→ The onload function is not run until all the information on the page is loaded. This leads to a substantial delay before any code is executed.

→ onDocumentReady loads the code just after the DOM is loaded. This allows early manipulation of the code.

46. How will you explain closures in JavaScript? When are they used?

→ Closure is a locally declared variable related to a function which stays in memory when the function has returned.

Ex:

```
function greet(message) {
    console.log(message);
}
function greeter(name, age) {
    return name + " says howdy!! He is " + age + " years old";
}
// Generate the message
var message = greeter("James", 23);
// Pass it explicitly to greet
greet(message);

Output:
//James says howdy!! He is 23 years old
```

This function can be better represented by using closures

```
function greeter(name, age) {
    var message = name + " says howdy!! He is " + age + " years old";
    return function greet() {
        console.log(message);
    };
}

// Generate the closure
var JamesGreeter = greeter("Jhon", 23);
// Use the closure
JamesGreeter();

//Result: Jhon says howdy!! He is 23 years old
```

47. How can a value be appended to an array?

→ A value can be appended to an array in the given manner -

```
arr[arr.length] = value;
```

48. What is the difference between .call() and .apply()?

→ The function .call() and .apply() are very similar in their usage except a little difference. .call() is used when the number of the function's arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.

→ The basic difference between .call() and .apply() is in the way arguments are passed to the function. Their usage can be illustrated by the given example.

```
var someObject = {
    myProperty : 'Foo',
    myMethod : function(prefix, postfix) {
        alert(prefix + this.myProperty + postfix);
    }
};

someObject.myMethod('<', '>'); // alerts '<Foo>'

var someOtherObject  = {
    myProperty : 'Bar'
};
someObject.myMethod.call(someOtherObject, '<', '>'); // alerts '<Bar>'
someObject.myMethod.apply(someOtherObject, ['<', '>']); // alerts '<Bar>'
```

49. Define event bubbling?

→ JavaScript allows DOM elements to be nested inside each other. In such a case, if the handler of the child is clicked, the handler of parent will also work as if it were clicked too.

50. Is JavaScript case sensitive? Give an example?

→ Yes, JavaScript is case sensitive. For example, a function parseInt is not same as the function Parseint.

51. What boolean operators can be used in JavaScript?

→ The 'And' Operator (&&), 'Or' Operator (||) and the 'Not' Operator (!) can be used in JavaScript. *Operators are without the parenthesis.

52. How can a particular frame be targeted, from a hyperlink, in JavaScript?

→ This can be done by including the name of the required frame in the hyperlink using the 'target' attribute.

```
<a href="/newpage.htm" target="newframe">>New Page</a>
```

53. Write the point of difference between web-garden and a web-farm?

→ Both web-garden and web-farm are web hosting systems. The only difference is that web-garden is a setup that includes many processors in a single server while web-farm is a larger setup that uses more than one server.

54. How are DOM utilized in JavaScript?

→ DOM stands for Document Object Model and is responsible for how various objects in a document interact with each other.

→ DOM is required for developing web pages, which includes objects like paragraph, links, etc. These objects can be operated to include actions like add or delete.

→ DOM is also required to add extra capabilities to a web page. On top of that, the use of API gives an advantage over other existing models.

55. How are event handlers utilized in JavaScript?

→ Events are the actions that result from activities, such as clicking a link or filling a form, by the user. An event handler is required to manage proper execution of all these events.

→ Event handlers are an extra attribute of the object. This attribute includes event's name and the action taken if the event takes place.

56. Explain the role of deferred scripts in JavaScript?

→ By default, the parsing of the HTML code, during page loading, is paused until the script has not stopped executing. It means, if the server is slow or the script is particularly heavy, then the webpage is displayed with a delay.

→ While using Deferred, scripts delays execution of the script till the time HTML parser is running. This reduces the loading time of web pages and they get displayed faster.

57. What are the various functional components in JavaScript?

The different functional components in JavaScript are-

→ *First-class functions:* Functions in JavaScript are utilized as first class objects. This usually means that these functions can be passed as arguments to other functions, returned as values from other functions, assigned to variables or can also be stored in data structures.

→ *Nested functions:* The functions, which are defined inside other functions, are called Nested functions. They are called 'everytime' the main function is invoked.

58. What are Screen objects?

→ Screen objects are used to read the information from the client's screen. The properties of screen objects are -

- → AvailHeight: Gives the height of client's screen
- → AvailWidth: Gives the width of client's screen.
- → ColorDepth: Gives the bit depth of images on the client's screen
- → Height: Gives the total height of the client's screen, including the taskbar
- → Width: Gives the total width of the client's screen, including the taskbar

59. What are the decodeURI() and encodeURI()?

→ EncodeURl() is used to convert URL into their hex coding. And DecodeURI() is used to convert the encoded URL back to normal.

```
<script>
    var uri="my test.asp?name=ståle&car=saab";
    document.write(encodeURI(uri)+ "<br>");
    document.write(decodeURI(uri));
</script>
```

Output -

```
my%20test.asp?name=st%C4%86%EF%BF%BDle&car=saab
my test.asp?name=stĆ�le&car=saab
```

60. Why it is not advised to use innerHTML in JavaScript?

→ innerHTML content is refreshed every time and thus is slower.

→ There is no scope for validation in innerHTML and, therefore, it is easier to insert rouge code in the document and, thus, make the web page unstable.

61. What does the following statement declares?

var myArray = [[[]]];

It declares a three dimensional array.

62. How are JavaScript and ECMA Script related?

→ ECMA Script are like rules and guideline while Javascript is a scripting language used for web development.

63. What is namespacing in JavaScript and how is it used?

→ Namespacing is used for grouping the desired functions, variables etc. under a unique name. It is a name that has been attached to the desired functions, objects and properties. This improves modularity in the coding and enables code reuse.

64. How can JavaScript codes be hidden from old browsers that don't support JavaScript?

→ For hiding JavaScript codes from old browsers:

Add "<!--" without the quotes in the code just after the <script> tag.
Add "//-->" without the quotes in the code just before the <script> tag.

Old browsers will now treat this JavaScript code as a long HTML comment. While, a browser that supports JavaScript, will take the "<!--" and "//-->" as one-line comments.