**JavaScript** (JS) is a programming language that is used both on the client-side and server-side that allows you to create dynamic and interactive web pages. used to implement behavior.

JavaScript code can be run inside of a browser, or in Node. Browsers and Node provide a runtime environment for our JavaScript code.

EcmaScript is a specification defined many new features for JavaScript.

<script> tag should be written inside <body> at bottom. Use src attribute for the .js file reference.

**Let, const and var difference:**

let having block scope and can be updated but cannot be redeclared.

var having global scope and can be updated and redeclared.

const cannot be redeclared and updated.

**Data Types:** number,string,boolean,bigint,null,undefined,object(object,array,date)

**typeof operator** to find the data type of a JavaScript variable.

The data type of null is object.

undefined and null are equal in value but different in type.

null === undefined // false

null == undefined // true

Conversion:

Conversion to number: Number(var\_name);

// 33 => 33

//33abc => NaN

//true=>1

Conversion to boolean: Boolean(var\_name);

//1 => true

//””=> false

// “asdfg” => true

If string is first then all considered as string. If last then first complete calculation.

Eg. 2+3+”5”=55 and “5”+2+3=523

// +true=1 & true+ = error

== check only value while === check both value and data type.

Primitive data type hold stack memory copy and non-primitive data type hold heap memory reference.

toString(): used to convert number to string.

toFixed(2):It will show 2 digits after decimal.

toPrecision(3):Showed 3 precision digit value.

toLocaleString(‘en-IN’): show number in Indian digit system.

Math():

Math.abs(-4):return positive value.

Math.round(4.3): round-off value. //4

Math.ceil(4.2): top round off value //5

Math.floor(4.9): low round off value //4

Math.random(): return random value between 0 to 1.

**Date** : new built-in object that stores the date, time and provides methods for date/time management.

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

Months are counted from zero (January is a zero month).

Days of week in getDay() are also counted from zero (that’s Sunday).

Date Get Methods:

getFullYear() Get year as a four digit number (yyyy)

getMonth() Get month as a number (0-11)

getDate() Get day as a number (1-31)

getDay() Get weekday as a number (0-6)

getHours() Get hour (0-23)

getMinutes() Get minute (0-59)

getSeconds() Get second (0-59)

getMilliseconds() Get millisecond (0-999)

getTime() Get time (milliseconds since January 1, 1970)

Date.now() returns the number of milliseconds since January 1, 1970.

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object. example setFullYear().

Array: js array while copy operation creates shallow copy.

Shallow copy: copy of object whose properties share the same reference point. Changes will reflect in original.

Deep copy : do not share the same reference.

Array Methods:

array.length

array.sort();

array.includes(‘element’); //check whether element is present or not and returns Boolean value.

array.push('new\_element'); // add element to last.

array.pop(); //removes last element.

array.shift(); remove first element.

array.unshift('');//add item at start.

array.reverse(); // reverse order of array elements.

Array.isArray(array\_name); // check whether object is array or not.

arr1.concat(arr2); // return new array with concatenation of elements.

array.indexOf('element'); // get index of element.

The slice() method can be used to create a shallow copy of an array or return a portion of an array. It does not alter the original array.

Array.slice(start index, end index); // end index item is excluded.

splice() method will change the contents of the original array. The splice() method is used to add or remove elements of an existing array and the return value will be the removed items from the array.

splice(starting index, optional delete count, optional items to add)

spread operator(…) is used to return all element.

JS Object:

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

If object created using constructor then it is singleton object.

If object created using literal then it is NOT singleton object.

Object Syntax(using object literals): const object\_name = {

key1: value1,

key2: value2

}

Accessing Object Properties:

objectName.key

objectName["propertyName"]

The Object.freeze() static method freezes an object. A frozen object can no longer be changed.

Object.freeze(obj\_name);

Combine objects:

1)const obj3=Object.assign({},obj1,obj2) //obj1,obj2 src obj and {} is target

Here The Object.assign() static method copies all properties from one or more source objects to a target object. It returns the modified target object. return target and target object are same.

2)spread operator.{…obj1,…obj2}

Object.keys(obj name): returns an array of a given object's string-keyed property names.

Object.values(obj name):returns an array of a given object' string-keyed property values.

Object.hasOwnProperty(‘property-name’): returns true if the specified object has the indicated property.

object destructuring : provides an alternative way to assign properties of an object to variables.

const object={ key1:val1, key2:val2}

let { key1: variable1, key2: variable2 } = object; //object destructuring

console.log(variable1) //val1

If the variables have the same names as the properties of the object, you can make the code more concise as follows:

let { key1, key2 } = object;

Function:

function functionName(parameters) {

// function body

}

functionName(arguments); //function calling or invoking.

If return is used then we can store function return value in a variable.

Every function in JavaScript implicitly returns undefined unless you explicitly specify a return value.

When declaring a function, you specify the parameters. However, when calling a function, you pass the arguments.

//Storing functions in variables we don’t execute the function but reference the function.

//i.e const result= add ;

The **function hoisting** allows you to call a function before declaring it. If function store in a variable then hoisting not allowed.

Rest operator(…) are used to access the multiple arguments .

We can pass array or object as arguments.

**Scope** determines the accessibility of variables.

Variables declared within a JavaScript function, are **LOCAL** to the function. Local variables have Function Scope.

Variables declared Globally (outside any function) have Global Scope.

**this keyword:**

this is used to refer current context.

If we call this by globally, not within a function, object, or whatever, it will refer to the global window object.

if we call this within an object method, we can use this to access properties and methods from the same object.

this within a function refer to the general window object.

We cannot access variable declared inside function using this.variable\_name as it will return undefined.

Example:

Function A () {

Let name=’vrish’;

console.log(this.name) //undefined

}

if you are using ES6 Modules, **strict** **mode** is automatically enabled, which will return **undefined** if using this in Global Scope.

In arrow function if we use curly braces then use return keyword. If () then no need of return.

**IIFE(Immediately Invoked Function Expression):**

It is a function defined as an expression and executed immediately after creation. One way to prevent the functions and variables from polluting the global object .

When you define a function and variable outside function the JavaScript engine adds the function to the global object.

Syntax: ( function (para) { //... } ) (arg);

False Value: false, 0,-0,0n,””,null,NaN,undefined

True Value:”0”, “ “,’false’,[],{},function(){}

**Closure function:**

A closure in JavaScript is a function that has access to the variables and parameters of its outer function, even after the outer function has returned.

Remember, inner function does not keep the separate copy of outer variables but it reference outer variables, that means value of the outer variables will be changed if you change it using inner function.

Closure is valid in multiple levels of inner functions.

function OuterFunction() {

var outerVariable = 100;

function InnerFunction() {

alert(outerVariable);

}

return InnerFunction;

}

var innerFunc = OuterFunction();

innerFunc(); // 100

**Nullish coalescing operator '??'**

returns the first argument if it’s not null/undefined. Otherwise, the second one.

var Val;

val=10??20 ;

console.log(val); //10

Val=null??10 //return 10 as first value is null.

break and continue in for() loop:

break the control flow of for() loop.

continue keywords skip the condition and continue the control flow.

**Map in Javascript:**

A Map holds key-value pairs where the keys can be any datatype.

Object Map

Not directly iterable Directly iterable

Do not have a size property Have a size property.

Keys must be Strings (or Symbols) Keys can be any datatype.

Keys are not well ordered. Keys are ordered by insertion

Have default keys Do not have default keys .

Syntax: const fruits = new Map ([ ["apples", 500], ["bananas", 300],["oranges", 200] ]);

set() Sets the value for a key or add element in a Map. fruits.set("apples", 500)

get() Gets the value for a key in a Map. fruits.get("apples");

size returns size of map. fruits.size

clear() Removes all the elements from a Map. fruits.clear();

delete() Removes a Map element specified by a key. fruits.delete("apples");

has() Returns true if a key exists in a Map. fruits.has("apples");

entries() Returns an iterator object with the [key, value] pairs in a Map.

keys() Returns an iterator object with the keys in a Map.

values() Returns an iterator object of the values in a Map.

**for…of loop:**

Use the for...of loop to iterate over elements of data structures such as Arrays, Strings, Maps.

For strings it return each character.

for (variable of iterable) {

// ...

}

We cannot use for…of loop on object as it will return Obj is not iterable error.

**For…in loop:**

Use the for...in loop to iterate over index of array and object.

To access value of object or array used *objectname /arrayname.[key]*

We cannot use on map object.

**forEach():**

JavaScript Array provides the forEach() method that allows you to run a function on every element.

It doesn’t return any value. If we tried to store in variable. It returns undefined.

Array.forEach( (val,index,arr)=>{});

We cannot use it on object.

The **some()** method returns true if any element in the array satisfies the condition.

**every()** method returns true if all elements in the array satisfy the condition.

**filter():**

used to extract elements from an array based on a condition and create a new array containing only the elements that satisfy that condition. return array size is less than or equal to original array.

filter() method can store result in new variable. You can chain the result with other array methods such as sort() and map().

*const newArray=oldArray.filter( (val)=> val>2)* //if val>2 write in {} then use return.

reduce():

allows you to reduce an array to a single value.

array.reduce(callbackFn [, initialValue]) //if initial value not passed then it is array[0]

function callbackFn(previousValue, currentValue, currentIndex, array)

**Array.map() :**

modified an array by applying a function to each element and return a new array with the modified values. return array size is exactly same as original array.

*const newArray = oldArray.map( (val)=> val\*10 )*

DOM: Document Object Model is an application programming interface (API) for manipulating HTML documents.

The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allow you to add, remove, and modify parts of the document effectively.

An **HTMLCollection** is a collection of document elements and can be accessed by their name, id, or index. We convert HTMLcollection to array using Array.from(collection\_name).

The getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.

A **NodeList** is a collection of document nodes can only be accessed by their index. index started from 0.

We can access each nodelist item using forEach(). To access single item use *nodelist[index].*

The querySelectorAll() method returns a static NodeList.

**innerHTML**: returns all text, including html tags.

**innerText**: returns the text as it appears on screen.

**textContent**: ignores all HTML tags and returns only the text. s

**document.getElementById()** returns a DOM element specified by an id.

The **querySelector()** method returns the first element that matches a specified selector.

To return all matches (not only first), use the **querySelectorAll()** instead.

Both querySelector() and querySelectorAll() throw a SYNTAX\_ERR exception if the selector(s) is invalid.

**Nodes** are element nodes, text nodes, and comment nodes. Whitespace between elements are also text nodes.

**Elements** are only element nodes.

The **children** property returns a collection of child elements.

**childNodes** returns NodeList of a child node (element nodes, text nodes, and comment nodes).

The **firstChild** and **lastChild** return the first and last child of a node, which can be any node type including text node, comment node, and element node.

The **firstElementChild** and **lastElementChild** return the first and last child Element node.

**nextElementSibling**: get the next sibling of current element.

**previousElementSibling :** get the previous siblings.

parentNode:

parentElement:

The **document.createElement()** creates a new HTML element.

Example const para = document.createElement("p"); //create a paragraph

The **setAttribute()** method sets a new value to an attribute.

element.setAttribute(name, value)

**createTextNode():**Creates a new Text node.

The **appendChild()** method to add a node to the end of the list.

**Event**:

**Events** are the occurrence of actions that can trigger certain functionality.

An **event** **handler** is a function with a block of code that is executed when a specific event fires.

**Event Listeners**: It listens to the event and executes when the event occurs.

event listener methods: addEventListener() and removeEventListener().

addEventListener('click', display, useCapture(t/f)) ; //click=>action display: function()

useCapture controls the phase of the propagation. passing true will cause the listener to be on the capturing phase. The default is false, which will apply it to the bubbling phase.

**Propagation** refers to how events travel through the Document Object Model (DOM) tree.

Bubbling and Capturing are the two phases of propagation.

**Bubbling** travels from the target to the root (flows upward).

**Capturing** travels from the root to the target (flows downward).

The target is the DOM node on which you click. The root is the highest-level parent of the target(document).

The **event object** is essentially the argument passed into the callback/event handler function. It provides information about the event, such as the target element, the type of event, etc.

**preventDefault()** provides way to prevent default behavior triggered in response to that event.

The **stopPropagation()** method immediately stops the flow of an event through the DOM tree.

**synchronous** code will block further execution of the remaining code until it finishes the current one.

**Asynchronous** code allows the program to be executed immediately.

**setTimeout**() method sets a timer which executes a function once the timer expires.

*setTimeout(function Ref, delay, param1, param2, /\* …, \*/ paramN)*

Timeouts are cancelled using **clearTimeout**().

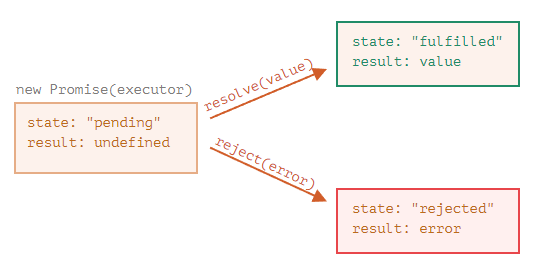
The **setInterval**() method continues calling the function at specified intervals until clearInterval() is called, or the window is closed.

*let promise = new Promise(function(resolve, reject) {*

*//Promise is a class*

*// executor : the executor runs automatically and attempts to perform a job.* *it calls resolve if it was successful or reject if there was an error.*

*});*

**

The promise object returned by the new Promise constructor is executed the function automatically when the promise is constructed.

If the asynchronous operation completes successfully, the executor will call the **resolve()** function to change the state of the promise from pending to fulfilled with a value.

In case of an error, the executor will call the **reject()** function to change the state of the promise from pending to rejected with the error reason.

To get the value of a promise when it’s fulfilled, you call the .**then()** method of the promise object.

If you want to get the error only when the state of the promise is rejected, you can use the **catch()** method of the Promise object.

We use try-catch() to handle error in async and await.

To execute the same piece of code whether the promise is fulfilled or rejected we use **.finally().**

**async and await** make promises easier to write.

The keyword async before a function makes ensure that the function return a promise.

The await keyword pause the execution of async function until the promise returns a result.

**fetch():**

method starts the process of fetching a resource from the network by making HTTP requests to servers, returning a promise that is fulfilled once the response is available.

A fetch() promise only rejects when a network error is encountered. A fetch() promise does not reject on HTTP errors (404, etc.)

fetch has high priority callback function stored in micro task queue that pass to call stack for execution.

**JavaScript Constructor Function:**

In JavaScript, a constructor function is used to **create objects**. To create an object from a constructor function, we use the **new keyword.**

The constructor function is useful if you want to create **multiple objects**. Object Literal is generally used to create a single object.

function Person () {

this.name = 'John'

}

const person = new Person();

You can also add properties and methods to a constructor function using a **prototype**.

Person.prototype.gender = 'Male';

**Prototype:**

In JavaScript, a prototype can be used to add properties and methods to a constructor function. And objects inherit properties and methods from a prototype.

In JavaScript, objects can inherit features from one another via prototypes. Every object has its own property called a prototype.

Because the prototype itself is also another object, the prototype has its own prototype. This creates a something called prototype chain. The prototype chain ends when a prototype has null for its own prototype.

Inheritance allows an object to use the properties and methods of another object.

JavaScript uses the prototypal inheritance.

In **prototypal inheritance**, an object “inherits” properties from another object via the prototype linkage.

If a prototype value is changed, then all the new objects will have the changed property value. All the previously created objects will have the previous value.

If an object tries to access the same property that is in the constructor function and the prototype object, the object takes the property from the constructor function.

Constructor\_function.prototype={age:50}

A standard way to implement prototypal inheritance in ES5:

***Object.create(proto, [propertiesObject])***

The Object.create() method creates a new object and uses an existing object as a prototype of the new object:

let A = Object.create(B); //creates an empty A object with the \_\_proto\_\_ of the B object

A.\_\_ proto\_\_ = B; If you want the object A to access all methods and properties of the object B

obj A={

name:’vrishabh’,

\_\_proto\_\_:B // B is object whose all properties can be accessed by obj A

}

Object.setPrototypeOf(B, A);// B object can access all properties of A

**Class**: A class is a blueprint for the object. You can create an object from the class.

// creating a class

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hello ${this.name}`);

}

}

let person1 = new Person(‘john’);

Classes always follow 'use-strict'. All the code inside the class is automatically in strict mode.

**Class Inheritance**:

Using class inheritance, a class can inherit all the methods and properties of another class.

To use class inheritance, you use the **extends** keyword.

Call(): Used to invoke a function with a specified this context, and optional arguments.

The call() method returns the result of calling the functionName().

Syntax: functionName.call(this, arg1, ... argN)

**Overriding Method** or Property

If a child class has the same method or property name as that of the parent class, it will use the method and property of the child class. This concept is called method overriding.

The **Browser Object Model (BOM)** is used to interact with the browser. The default object of browser is window. We can call methods and properties of window object to interact with browser directly.

Window properties:

*innerWidth* and *innerHeight* properties return the size of the page viewport inside the browser window (not including the borders and toolbars).

The *outerWidth* and *outerHeight* properties return the size of the browser window itself.

window.open To open a new window or tab, you use the window.open() method

window.open(url, windowName, [windowFeatures]);

To resize a window you use the resizeTo() method of the window object

window.resizeTo(width,height):

To close a window, you use the window.close() method.

alert() method to display information that you want users to acknowledge.

The confirm() shows a system dialog that consists of a question and two buttons: OK and Cancel.

The prompt() shows a dialog that prompts the user to enter a text and wait until the user submit or cancel the dialog.

Call: change the context of invoking function./replace value of this. We can pass n number of arguments.

Apply: change the context of invoking function./replace value of this. We can pass n number of arguments but in ARRAY.

Bind return copy of function that can we store in variable and we can use it.