**JavaScript Interview Questions**

**What is DOM?**

DOM is nothing but a Document Object Model.

* It is a programming interface for HTML and XML.
  + It represent the page that program can change the structure style and content.
* That way programming languages connect to the web page.
* The HTML DOM can be accessed with JavaScript.
* JavaScript can change all the HTML elements in the page
* JavaScript can change all the HTML attributes in the page
* JavaScript can change all the CSS styles in the page
* JavaScript can remove existing HTML elements and attributes
* JavaScript can add new HTML elements and attributes

**DOM Methods:**

* innerHTML

**What is OnLoad?**

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

**What is BOM?**

The BOM allows JavaScript to “interact with” the browser. The way of talking will do in browser.

**What is dependency and devdependency?**

**Ans:** The difference between these two, is that devDependencies are modules which are only required during development, while dependencies are modules which are also required at runtime.

**Dependency**: npm install JavaScript --save or npm install JavaScript –S

**Devdependency:** npm install JavaScript --save -dev or npm install JavaScript –D

**What is the Use of JavaScript HTML methods is getElementById ()?**

**Ans:**  This example uses the method to "find" an HTML element (with id="demo") and changes the element content (innerHTML) to "Hello JavaScript".

**Ex:** document.getElementById (“demo”).innerHTML= “Hello Sai”

**What is the Function?**

**Ans:** A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

**Ex:** function myFunction(p1, p2) {  
   return p1 \* p2;

}

**First Order Function**

JavaScript regular functions are also First-Order Functions

It never take the functions are one of its Parameters and doesn’t return the another Functions.

**High Order Functions**

## A ****higher-order****function is a function that either takes a function as one of its parameters or returns another function.

there are several other higher-order functions: **.map,** .**filter**, .**reduce**, etc.

**With Higher-order function map**

const arr1 = [1, 2, 3];

function myMapFunc(item) {  
 return item \* 2;  
}

const arr2 = arr1.map(myMapFunc);

console.log(arr2); => o/p => 2,4,6

We can make this even shorter using the arrow function syntax:

const arr1 = [1, 2, 3];

const arr2 = arr1.map(item => item \* 2);

console.log(arr2);

**Without Higher-order function**

const arr1 = [1, 2, 3];  
const arr2 = [];

for(let i = 0; i < arr1.length; i++) {  
 arr2.push(arr1[i] \* 2);  
}

console.log(arr2); => OutPut => [ 2, 4, 6 ]

**Using Filter Higher Order Function**

var ages = [32, 33, 16, 40];

function checkAdult(age) {  
  return age >= 18;  
}  
  
function myFunction() {  
   document.getElementById("demo").innerHTML =

ages.filter(checkAdult);

}

**What is Splice Method in Arrays?**

The splice() method adds/removes items to/from an array, and returns the removed item(s).

**Example:**

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

document.write("Original Array:<br>" + fruits);

function myFunction() {

fruits.splice(2, 1, "Lemon", "Kiwi");

document.write("New Array:<br>" + fruits);

}

**OutPut => New Array: Banana,Orange,Lemon,Kiwi ,Mango**

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

**What is the slice()?**

**Ans:** slice() extracts a part of a string and returns the extracted part in a new string.

It takes the values from 1 to soon.

The method takes 2 parameters: the start position, and the end position (end not included).

**Ex:** var str = "Apple, Banana, Kiwi";  
var res = str.slice(7, 13);

**The result of res will be:** Banana

var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12, -6)

**The result of res will be:** Banana

**Note:** In slice Method it can include comas and white spaces. (2nd parameter) It counted to starting point.

If a parameter is negative, the position is counted from the end of the string.

**What is the subString() ?**

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

**What is the subStr() ?**

substr() is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

**Ex:** var str = "Apple, Banana, Kiwi";  
var res = str.substr(7, 5);

**The result of res will be:** Banan

**Note:** If the first parameter is negative, the position counts from the end of the string.

**What is charAt()?**

The charAt() method returns the character at a specified index (position) in a string:

**Example:** var str = "HELLO WORLD";  
str.charAt(0);     // returns H

**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.

**What is the use of blur function?**

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

**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.

**What is Call And Apply Methods?**

**Call Method**

The **call()** method calls a function with a given **this** value and arguments provided individually.

The call() method is a predefined JavaScript method.

It can be used to invoke (call) a method with an owner object as an argument (parameter).

### Example:-

var person = {  
  **fullName**: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
var person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}

var person2 = {  
  firstName:"Mary",  
  lastName: "Doe"  
}  
person.fullName.call(**person1**);  // Will return "John Doe"

**Apply Method**

The apply() method is similar to the call() method (previous chapter).

In this example the **fullName** method of **person** is **applied** on **person1**:

### Example

var person = {  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
var person1 = {  
  firstName: "Mary",  
  lastName: "Doe"  
}  
person.fullName.apply(person1);  // Will return "Mary Doe"

## The Difference Between call() and apply()

The difference is:

The call() method takes arguments **separately**.

The apply() method takes arguments as an **array**.

The apply() method is very handy if you want to use an array instead of an argument list.

## The apply() Method with Arguments

The apply() method accepts arguments in an array:

### Example

var person = {  
  fullName: function(city, country) {  
    return this.firstName + " " + this.lastName + "," + city + "," + country;  
  }  
}  
var person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
person.fullName.apply(person1, ["Oslo", "Norway"]);

**Compared with the call() method:**

### Example

var person = {  
  fullName: function(city, country) {  
    return this.firstName + " " + this.lastName + "," + city + "," + country;  
  }  
}  
var person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
person.fullName.call(person1, "Oslo", "Norway");

## Using Math.max() on an Array

You can use Math.max.apply to find the highest number in an array:

### Example

<script>

var points = [40, 100, 1, 5, 25, 10];

document.getElementById("demo").innerHTML = myArrayMax(points);

function myArrayMax(arr) {

return Math.max.apply(null, arr);

}

</script>  
**Sorting**

**Sorting an array :-**

<script>

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

document.getElementById("demo").innerHTML = fruits;

function myFunction() {

fruits.sort();

document.getElementById("demo").innerHTML = fruits;

}

</script>

**Math Sorting ascending Order**

var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});

O/P = 1,5,10,25,40,100

**Math Sorting descending Order**

var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b - a});

O/P = 100,40,25,10,5,1

## Array.forEach()

The forEach() method calls a function (a callback function) once for each array element.

### Example

### var txt = "";

### var numbers = [45, 4, 9, 16, 25];

### numbers.forEach(myFunction);

### document.getElementById("demo").innerHTML = txt;

### function myFunction(value) {

### txt += value + "<br>";

### }

## Array.map()

The map() method creates a new array by performing a function on each array element.

 we want to update the value of an array of objects using its key.

By using the [JavaScript array map() function](https://codesquery.com/javascript-map-function/) we can easily do that.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

#### .map() have some parameters

#### currentValue

The current element being processed in the array.

#### index

The index of the current element being processed in the array.

        var arr = [2, 5, 6];

        var newArr = arr.map(function(val, index){

            return {key:index, value:val\*val};

        })

        console.log(newArr) 🡺 key: 0, value:4

key: 1, value: 25

key: 2, value: 36

Example:

var numbers1 = [45, 4, 9, 16, 25];  
var numbers2 = numbers1.map(myFunction);

document.write(numbers2);  
  
function myFunction(value) {  
  return value \* 2;  
}

**What is the Difference Between .forEach() and .map() methods**

**map():** creates a new array with the results of calling a provided function on every element in this array.

**forEach():**  executes a provided function once for each array element.

Now let’s understand the meaning of the above two statements. [map() method](https://codesquery.com/javascript-map-function/) calls a provided function on each element of the provided array and **returns a new array of the same size or new size** while [**forEach() method**](https://codesquery.com/javascript-foreach-function/)**doesn’t return anything**, its simple calls the provided function on each element of the array.

## Array.filter()

The filter() method creates a new array with array elements that passes a particular test.

This example creates a new array from elements with a value larger than 18:

### Example

var numbers = [45, 4, 9, 16, 25];  
var over18 = numbers.filter(myFunction);  
  
function myFunction(value) {  
  return value > 18;  
}

**To Find The Day (Years count as an array)**

Var d = new Date(2019, 06,20):

O/P = sat 20-07-2019

## Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

### Example

Math.round(4.7);    // returns 5  
Math.round(4.4);    // returns 4

## Math.ceil()

Math.ceil(x) returns the value of x rounded **up** to its nearest integer:

### Example

Math.ceil(4.4);     // returns 5

## Math.floor()

Math.floor(x) returns the value of x rounded **down** to its nearest integer:

### Example

Math.floor(4.7);    // returns 4

## The "use strict" Directive

use strict"; Defines that JavaScript code should be executed in "strict mode".

The "use strict" directive was new in ECMAScript version 5.

1. When we declared duplicate Variables and parameters ‘Use Strict’ throw an error
2. When we declared value then we can’t put a **Var** keyword declaration in front of that variable we can use that variable it will throw an error

The strict mode in JavaScript does **not** allow following things:

1. Use of undefined variables
2. Use of reserved keywords as variable or function name
3. Duplicate properties of an object
4. Duplicate parameters of function
5. Assign values to read-only properties
6. Modifying arguments object
7. Octal numeric literals
8. with statement
9. eval function to create a variable

## What is this?

The JavaScript this keyword refers to the object it belongs to.

It has different values depending on where it is used:

* In a method, this refers to the **owner object**.
* Alone, this refers to the **global object**.
* In a function, this refers to the **global object**.
* In a function, in strict mode, this is undefined.
* In an event, this refers to the **element** that received the event.
* Methods like call(), and apply() can refer this to **any object**.

Example :

<script>

var person = { firstName: "John", lastName : "Doe", id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

document.getElementById("demo").innerHTML = person.fullName();

</script>

## Event Bubbling or Event Capturing?

# Event Bubbling :

Event Bubbling is the event starts from the **deepest element or Largest element to its parents, then all its ancestors which are on the way to bottom to top**. At present, all the modern browsers have event bubbling as the default way of event flow.

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

**If you want to stop the event bubbling, this can be achieved by the use of the event.stopPropagation() method.**

<button id="child" onclick="event.stopPropagation()">Child</button>

Example:

|  |
| --- |
| <div id="parent"> |
|  | <button id="child">Child</button> |
|  | </div> |
|  |  |
|  | <script> |
|  | var parent = document.querySelector('#parent'); |
|  | <!-- Add click event on parent div --> |
|  | parent.addEventListener('click', function(){ |
|  | console.log("Parent clicked"); |
|  | }); |
|  |  |
|  | var child = document.querySelector('#child'); |
|  | <!-- Add click event on child button --> |
|  | child.addEventListener('click', function(){ |
|  | console.log("Child clicked"); |
|  | }); |
|  | </script> |

# Event Capturing:

**Event Capturing** is the event starts from top element to target element. Modern browser doesn’t support event capturing by default but we can achieve that by code in JavaScript.

In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

**Example:**

|  |
| --- |
| <div id="parent"> |
|  | <button id="child">Child</button> |
|  | </div> |
|  |  |
|  | <script> |
|  | var parent = document.querySelector('#parent'); |
|  | var child = document.querySelector('#child'); |
|  |  |
|  | parent.addEventListener('click', function(){ |
|  | console.log("Parent clicked"); |
|  | },**true**); |
|  |  |
|  |  |
|  | child.addEventListener('click', function(){ |
|  | console.log("Child clicked"); |
|  | }); |

**What is a “closure” in JavaScript? Provide an example**

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

The closure has access to variables in three scopes:

* Variables declared in their own scope
* Variables declared in a parent function scope
* Variables declared in the global namespace

**Example:**

**function init() {**

**var name = 'Mozilla';**

**function displayName() { closure**

**alert(name);**

**}**

**displayName();**

**}**

**init();**

**What is Hoisting?**

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code is execution.

In JavaScript, Hoisting is the default behaviour of moving all the declarations at the top of the scope before code execution.

### Example

x = 5;   
elem = document.getElementById("demo");

elem.innerHTML = x;    
var x;

**What is Constructor?**

The constructor method is a special method for creating and initializing an object created within a [class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class).

class Polygon {

constructor() {

this.name = "Polygon";

}

}

var poly1 = new Polygon();

console.log(poly1.name);

// expected output: "Polygon"

**What is Var, Let and Const?**

**Var is function scope variable**

### Function Scope

Whenever you declare a variable in a function, the variable is visible only within the function. You can't access it outside the function. **var** is the keyword to define variable for a function-scope accessibility.

function foo(){

var fruit ='apple';

console.log('inside function: ',fruit);

}

foo(); //inside function: apple

console.log(fruit); //error: fruit is not define

**Let and Const Keyword**

Let and Const keywords are block scope variable

**Block-Scope**

A block is a set of opening and closing curly brackets.

A block scope is the area within **if**, **switch** conditions or **for** and **while** loops. Generally speaking, whenever you see **{curly brackets}**, it is a block. In ES6, **const** and **let** keywords allow developers to declare variables in the block scope, which means those variables exist only within the corresponding block.

**What is spread Operator?**

It allows the array expression to be expanded to 0 or more arguments are expected.

**or**

an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

Example:

Var arr = [1, 2, 3, 4]

Function myArr(x, y,z) {

return x\*y\*z

}

Console.log(myArr(…arr)); => Output : 24

## What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies are used to whatever the data sent the webserver that data stored and the connection is comes back that time it will send data to the browser.

## Create a Cookie with JavaScript

JavaScript can create, read, and delete cookies with the document.cookie property.

With JavaScript, a cookie can be created like this:

document.cookie = "username=John Doe";

**What is null and undefined?**

**Null:** null is typeof object when we assign the null to a current variable it doesn’t have any value it will have later. Finally null means it is absence of value.

**Example: null**

var myVar = null;

alert(myVar); // null

## null !== undefined

As you can see so far, null and undefined are different, but share some similarities. Thus, it makes sense that null does not strictly equal undefined.

null !== undefined

But, and this may surprise you, null loosely equals undefined.

null == undefined

**Undefined:** When declare a variable and we don’t assign any value to that variable and we can use that variable to display it returns **Undefined.**

Example:

Var s;

console.log(s); Output is Undefined;

**What are the data types in Javacsript?**

String

Number

Boolean

Null

Undefined

**JavaScript non-primitive data types**

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| **Object** | represents instance through which we can access members |
| **Array** | represents group of similar values |
| **RegExp** | represents regular expression |

**setInterval**

The setInterval() method calls a function expression at specified intervals (in milliseconds).

The setInterval() method will continue calling the function until [clearInterval()](https://www.w3schools.com/jsref/met_win_clearinterval.asp) is called, or the window is closed.

**Example**

<button click=”myFunction()” >Click Here</button>  
  
function myFunction() {  
  setInterval(alertFunc, 3000);  
}

function alertFunc() {  
  alert("Hello!");  
}

**setTimeOut**

The setTimeout() method calls a function expression after a specified number of milliseconds.

**Tip:** 1000 ms = 1 second.

**Tip:** The function is only executed once.

### Example

Display an alert box after 3 seconds (3000 milliseconds):

setTimeout(function(){ alert("Hello"); }, 3000);

**clearTimeout**

The clearTimeout() method clears a timer set with the [setTimeout()](https://www.w3schools.com/jsref/met_win_settimeout.asp) method.

**How to Add the objects in Array?**

arrObj = [

{ name: 'Krunal', age: 26 },

{name:'Ankit', age: 24 }

];

arrObj.push ({name: 'Rushabh', age: 27});

console.log (arrObj);

O/P = [

{ name: 'Krunal', age: 26 },

{name:'Ankit', age: 24 } ,

{name: 'Rushabh', age: 27}

];

**How to Add the object properties in Array?**

Var s = [

{ name: 'Krunal', age: 26 },

{name:'Ankit', age: 24 }

];

S[1].lastName = “Krishna”;

Console.log(s[1]. lastName) 🡺 Krishna

**How to display Multiple Object values?**

Var txt =” ”;

Var x;

var person = {fname:"John", lname:"Doe", age:25};  
  
for (x in person) {  
  txt += person[x];  
}

console.log(txt); 🡺 john doe 25

**How to delete the object property?**

var person = {

firstname:"John",

lastname:"Doe",

age:50,

eyecolor:"blue"

};

delete person.age;

document.getElementById("demo").innerHTML =

person.age + " years old."; 🡺 undefined

**What are the types of Objects?**

**1.Literals Notation**

**2.Constructor function using new keyword**

Objects created using object literals are singletons. This means when we change the object, it affects that object across the entire script.

Object defined with a function constructor let us have multiple instances of that object. This means change made to one instance, will not affect other instances.

**Create the Objects using Object.create() ?**

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

var s = Object.create(person);

s.firstName = "Sai",

s.lastName = "Krishna",

s.id = 4455,

alert( person.fullName());

alert(s.fullName());

**Add a Property in array of Objects?**

<script>

var person = [

{name: "Sai", design: " UI developer"},

{name: "Ram", design: "Java developer"},

{name: "Satwik", design: "Python developer"},

{name: "John", design: " UI developer"}

]

person.forEach(function(item) {

item.age = 220;

})

for(var i = 0; i < person.length; i++) {

person[i].age = 220;

}

document.write(person[2].age);

</script>

**Dynamically Add Div when user click the div**

<div onclick="myFunction()"></div>

<script>

function myFunction() {

var mydiv = document.createElement("div");

document.body.appendChild(mydiv);

}

</script>

## Prototype Inheritance

The **prototype** property allows to add new properties and methods to existing object types

All JavaScript objects inherit properties and methods from a prototype:

* Date objects inherit from Date.prototype
* Array objects inherit from Array.prototype
* Person objects inherit from Person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

**<script>**

**function Person(first, last, age, eye) {**

**this.firstName = first;**

**this.lastName = last;**

**this.age = age;**

**this.eyeColor = eye;**

**}**

**Person.prototype.nationality = "English";**

**var myFather = new Person("John", "Doe", 50, "blue");**

**document.getElementById("demo").innerHTML =**

**"The nationality of my father is " + myFather.nationality; 🡺 English**

**</script>**

**What are the Arrow Functions in JavaScript ES6?**

* + An **arrow function expression** is a syntactically compact alternative to a regular [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function),
  + The handling of this is also different in arrow functions compared to regular functions.
  + Arrow functions does not bind its own this.

**What is the use of isNaN function?**

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

**What is a prompt box?**

A prompt box is a box which allows the user to enter input by providing a text box.

**What is === operator?**

=== is called as strict equality operator which returns true when the two operands are having the same value and same type conversion.

**What is == operator?**

== is called equality operator which returns true when the operands are having the equal value and any type conversion.

**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.

**What is break and continue statements?**

Break statement exits from the current loop.

Continue statement continues with next statement of the loop.

## The While Loop

The while loop loops through a block of code as long as a specified condition is true.

Example:

var text = "";

var i = 0;

while (i < 10) {

text += i;

i++;

}

document.getElementById("demo").innerHTML = text; => 1 to 9

**do While Loop**

The loop will always be executed at least once, even if the condition is false,

because the code block is executed before the condition is tested:

### Example

do {  
  text += "The number is " + i;  
  i++;  
}  
while (i < 10);

**Find the output of given code**

var s = 1;

if(s = 2) {

document.write("s2 :" + s); Output => 2

}

function myFunc() {

var s = 3;

}

myFunc();

document.write("s3:" + s); Output => 2

function myFunction() {

var s = 4;

}

myFunction();

document.write("s4 :" + s); Output => 2

**Find the Output of given Code**

<button onclick="myFunction()">Try it</button>

<script>

function myFunction() {

setInterval(myFunc,0);

}

function myFunc() {

for(var i=0; i< 10; i++) {

document.write(i);

}

}

</script>

Output => 0123456789….0123456789…0123456789… continuously return the 0 to 9 digits in **Zero(0 sec)** time of intervals.

**What is ParseFloat()?**

The parseFloat() function parses a string and returns a floating point number.

**Note:** Only the first number in the string is returned!

**Note:** Leading and trailing spaces are allowed.

**Note:** If the first character cannot be converted to a number, parseFloat() returns NaN.

**Example:**

var a = parseFloat("10") 🡺 10

var b = parseFloat("10.00") 🡺10

var c = parseFloat("10.33") 🡺10.33

var d = parseFloat("34 45 66") 🡺 34

var e = parseFloat(" 60 ") 🡺60

var f = parseFloat("40 years") 🡺 40

var g = parseFloat("He was 40") 🡺 NaN

**What is ParseInt()?**

The parseInt() function parses a string and returns an integer.

**Note:** Only the first number in the string is returned!

**Note:** Leading and trailing spaces are allowed.

**Note:** If the first character cannot be converted to a number, parseInt() returns NaN.

**Example:**

var a = parseInt("10") 🡺 10

var b = parseInt("10.00") 🡺10

var c = parseInt("10.33") 🡺 10

var d = parseInt("34 45 66") 🡺34

var e = parseInt(" 60 ") 🡺60

var f = parseInt("40 years") 🡺40

var g = parseInt("He was 40") 🡺 NaN

# Instanceof operator in JavaScript

The **instanceof operator** in JavaScript is used to check the type of an object at run time. It returns a Boolean value if true then it indicates that the object is an instance of a particular class and if false then it is not.

**ES6 (ECMAScript)**

**ES6 Feature**

* Let and Const
* Arrow functions
* Default parameters
* for of loop
* Spread attributes
* New String Methods
* Maps
* Sets
* Static methods
* Getters and Setters

**Default parameters :-**

When we declare function with having the some parameters to that function and assign the some value to that parameters is called Default Parameters.

**Example : 1 🡺**

Let Func = (a, b=10) => {

return a+b;

}

Document.write(Func(20)) 🡺 Output 🡺 30

**Example : 2 🡺**

Let Func = (a = 10, b) => {

return a+b;

}

Document.write(Func(20)) 🡺 Output 🡺 NaN

We need to declare a function with parameters they get the values in order.

The first value get assigned to the first parameter and second value get assigned to the second parameter.

In the above example, the value 20 gets assigned to parameter ‘a’ and ‘b’ is not having any value.

**Spread Operator**

**Example without spread attributes:**

let SumElements = (arr) => {  
 console.log(arr); // [10, 20, 40, 60, 90]

let sum = 0;  
 for (let element of arr) {  
 sum += element;  
 }  
 console.log(sum); // 220.   
 }

**SumElements([10, 20, 40, 60, 90]);**

**Example with spread attributes:**

let SumElements = (...arr) => {  
 console.log(arr); // [10, 20, 40, 60, 90]

let sum = 0;  
 for (let element of arr) {  
 sum += element;  
 }  
 console.log(sum); // 220.   
}

**SumElements(10, 20, 40, 60, 90);**

**Note we are not passing array here. Instead we are passing the elements as arguments.**

**Concat the Two Arrays using Spread Operator**

Var s = [1,2,3,4]

Var T = [5,6,7,8]

Var L = […S,…T]

Document.write(L) 🡺 output 🡺 [1,2,3,4,5,6,7,8]

**Concat the Two Objects using Spread Operator**

var s = {

name: "sai",

age: 20

}

var t = {

lname: "koleti",

city: "hyd"

}

var l = {...s,...t}

console.log(l); 🡺 Output 🡺 {name: "sai", age: 20, lname: "koleti", city: "hyd"}

**Concat the two Objects Using Object.assign()**

Var tech = {

name: “sai”,

age: 25

}

Var app = {

designation: “UI Developer”,

City : “Hyd”

}

Var target = Object.assign(tech,app)

Console.log(target)

OutPut 🡺 = {name: “sai”, age: 25, designation: “UI Developer”.

City : “Hyd” }

**New String Methods**

The following String methods as follows 🡺

* repeat
* startsWith
* endsWith
* includs

Repeat() method 🡺

Var s = “goodbye ”;

Console.log(s.repeat(4)) 🡺 goodbye goodbye goodbye goodbye

StartsWith() 🡺

Var s = “goodbye ”;

Console.log(s.startsWith(“good”)) 🡺 true

Var s = “goodbye ”;

Console.log(s.startsWith(“bye”, 4)) 🡺 true

The second parameter indicates the starting position from “bye”

Var s = “goodbye ”;

Console.log(s.endsWith(“bye”)) 🡺 true

Var s = “goodbye ”;

Console.log(s.endsWith(“good”, str.length-3 )) 🡺 true

**What is the use of Set?**

The **Set** object lets you store unique values of any type, whether [primitive values](https://developer.mozilla.org/en-US/docs/Glossary/Primitive) or object references.

Example:-

const set1 = new Set([1, 2, 3, 4, 5]);

console.log(set1.has(1)); 🡺 true

console.log(set1.has(4)); 🡺 true

console.log(set1.has(7)); 🡺 false

console.log(set1.has(6)); 🡺 false

**How con cat the two arrays using spread operator and Delete the Duplicates using sets**

**var arr1 = [1,2,3,4,4,4];**

**var arr2 = [5,5,6,7,8,8];**

**var myList = [...arr1, ...arr2] 🡺 O/P 🡺 [1,2,3,4,4,4, 5,5,6,7,8,8]**

**var app = new Set(myList);**

**var myArray = [...app]**

**console.log(myArray); 🡺 [1,2,3,4,5,6,7,8]**

**How to remove the duplicate numbers in an Array?**

var s = [1,2,3,3,4,4,4,5,6,6];

var k = s.filter((item,index)=> s.indexOf(item)===index);

console.log(k);

output 🡺 [1, 2, 3, 4, 5, 6]

**OR**

var s = [1,2,3,3,4,4,4,5,6,6];

var k = new Set(s);

console.log(…k); 🡺 [1, 2, 3, 4, 5, 6]

**What are the Promises in JavaScript?**

**Promises** are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations associated With callback Functions.

**Example:-**

let promise = new Promise(function(resolve, rejet){

let value = true;

if(value) {

resolve(“This Promise is True”)

}else {

reject(“This Promise is False”)

}

})

promise.then(resolve => console.log(resolve),

reject => console.log(reject)

)

O/P => This Promise is True

**IndexOf:**

The indexOf() method returns the position of the first occurrence of a specified value in a string.

**Example:**

var str = "Hello world, welcome to the universe welcome.";

var n = str.indexOf("welcome"); Output 🡺 13

**lastIndexOf**

The lastIndexOf() method searches the array for the specified item, and returns its position. It returns second occurrence in the array

It returns Last index of the ARRAY

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

var a = fruits.lastIndexOf("Apple"); Output 🡺 2

Returns -1 if the item is not found.

It returns Last index of the ARRAY

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

var a = fruits.lastIndexOf("Apple"); Output 🡺 4

**JSON**

1. **What is JSON?**

* **JSON** stands for JavaScript Object Notation.
* JSON we can use to Exchanging and Storing the data browser and server.
* JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
* We can also convert any JSON received from the server into JavaScript objects.

## Advantages of JSON

* JSON is easy to read and write.
* It is a lightweight text-based interchange format.
* JSON is language independent.

**Storing Data**

myObj = {name: "John", age: 31, city: "New York"};  
myJSON = JSON.stringify(myObj);  
localStorage.setItem("testJSON", myJSON);

**Retrieving data:**text = localStorage.getItem("testJSON");  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;

# JSON.parse()

It will convert the JSON into JavaScript Object.

**Example:**

var txt = '{"name":"John", "age":30, "city":"New York"}'

var obj = JSON.parse(txt);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;

**JSON.stringify()**

It will convert the JavaScript Object into JSON String.

**Example:** var obj = { name: "John", age: 30, city: "New York" };  
var myJSON = JSON.stringify(obj);  
document.getElementById("demo").innerHTML = myJSON;