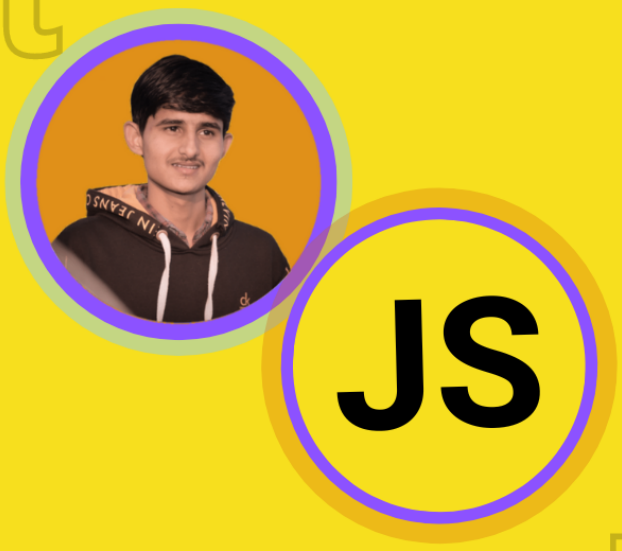
JavaScript Doctory

By raju_webdev



Choose Your Favorite Mentor

Nisha Singla

CodeWithHarshad

Mani

What is JavaScript?

- Most popular and widely used scripting language in the field of web development.
- It is an object-oriented language which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a scripting language or interpreted language.
- Used for client-side scripting and server-side scripting. And is case sensitive language.

Features of JavaScript:

- It is a light-weight and case sensitive language.
- JavaScript is an object-oriented language. So, it have the concept of classes, inheritance, etc.
- It provides a good control to the users over the web browsers.
- JavaScript is used for client-side & server-side scripting. So for Full-Stack Developer no need to learn another language to become a full-stack web developer.

Why choose JavaScript in your web development journey?

- JavaScript is very popular and the popularity of the JavaScript is increasing day-by-day.
- It is used in almost every field like web development, game development, mobile apps development, etc.
- It has a large number of support community with daily active users.
- For the development you don't need to setup extra environment or compiler for the execution of the code.

Companies who uses JavaScript

- Google
- Facebook
- Paypal
- Netflix
- Microsoft

Where JavaScript is used?

- JavaScript is used to build dynamic web pages and websites.
- It have the great libraries and frameworks to build powerful web and mobile applications.
- JavaScript is also used to create 2D and 3D games.
- It is also used to create web servers with node.js which is used to execute the javascript code.
- It is also used to create single page web applications.

How to run JavaScript code in browser?

- To run the javascript code in the browser just open you browser right click and select inspect then select console and you can write your code in the browser.

Example:

i. console is basically used to print the output in console of the browser.
```
console.log('Hello Web Developer Welcome to Javascript Doctory Series');
```

ii. alert is basically used to show the alert box on the browser window.
```
alert('Alert box got Javascript Doctory Series');
```

iii. prompt is used to get the value and store it to the variable. (we will learn about variables in javascript soon).
```
prompt('Promptt box got Javascript Doctory Series');
```

What is Variable?

- Variables are just like a container which we use to store some values.
- JavaScipt provides three types of keywords to declare varia bles.
- let, var, const are used to declare variables in javascript.
- In these day var is less used to declare variables.

Rules for creating variables:

- Variable name must start with letters(a to z or A to Z), underscore(_), dollar sign ($).
- Variables are case sensitive. So raju_webdve and Raju_Webdev are different.
- After first letter of variable name we can use number (digits) in variable name. Like let raju_webdev20;

Using variables:

➢ let

let keyword is used to declare variable in javascript. The value of variable can be changed which variable is created using let keyword.

Example:

```
let languageName = 'JavaScript';
console.log(languageName);
```

➢ var

var keyword is also used to declare variable in javascript. It has global because If you initialize the value of the variable then it will change the value. And you cannot declare the same name variable aging using var.

Example:

```
var myWebsite = 'geeks help';
{
  myWebsite = 'rajusheoran.me';
}
console.log(myWebsite);
```

➢ const

As the name suggest const the variable which is created using const cannot be changed the value can be declare and initialize once.

Example:

```
const name= 'raju webdev';
console.log(name);
```

What are Datatypes?

- In JavaScipt datatype define that which type of value can be stored in variables. And we can also find that what it the type of the variable using typeof(variable_name)

Types of DataType in JavaScirpt:

- In javascript there are mainly two types of data types. Premitive data types and Reference(derived) data types.

Using Datatypes:

➢ Using Premitive Data Types

Premitive data types are those data types who are already define in the javscript language. There is multiple type of primitive data types in javascript and these are:

i). String: It is used to store the value in string data type.
ii). Boolean: It is used to store the value Boolean. It can be true of false.
iii). Number: Number is used to store number type value.
iv). Null: It is used to represent an empty or unknown value.
v). Undefined: When we don't assing any value to the variable.

Example:

```
let name = 'raju webdev';
console.log(name, typeof(name));

let age = 20;
console.log(age, typeof(age));

let frontendDeveloper = true;
console.log(frontendDeveloper, typeof(frontendDeveloper));
```

```
let city;
console.log(city, typeof(city));

let phNumber = null;
console.log(phNumber, typeof(phNumber));
```

➢ Using Reference Data Types

Reference data types are those data types which are not predefine. And these data types are also known as derived data types. And the reference or derived data types in javascript are:

i).Object: It is used to store multiple collections of data in the pair or key value.

ii).Array: An array is an object that can store multiple values at once.

iii).Dates: It is mainly used to get and set dates in javascript.

iv).Functions: It is a block of code which perform a specific task.

Example:

```
let myObject = {
    name: 'raju_webdev',
    website: 'geeks help',
    role: 'frontend developer',
age: 20
}
console.log('Our object is: ' + myObject)

let myArray = [4, 15, 20, 22, 'geeks help'];
console.log('Your array is: ' + myArray);

let newDate = new Date();
console.log('Current Date is: ' + newDate);

function myFunction(){
console.log('This is myFunction');
}
```

```
myFunction();
```

String

- It is a collection of one or more than one characters.
- String can contain letters, symbols and numbers.
- It is a primitive data type.

String methdos:

- **Length**

  This string methods is used to find the length of the given string

  Example:

```
let name = 'geeks help';
console.log(name);
console.log('Length of string name is: ', name.length);
```

  output:

  geeks help

  Length of string name is:10

- **toLowerCase()**

  It is used to convert the string into lower case

  Example:

```
let name = 'GEEKS Help';
console.log(name)
console.log('String into lower case is: '+ name.toLowerCase());
```

  output:

  GEEKS Help

  String into lower case is: geeks help

- **toUpperCase()**

  It is used to convert the string into upperCase

  Example:

```
let name = 'geeks Help';
console.log(name)
console.log('String into upper case is: '+ name.toUpperCase());
```

output:

geeks Help

String into lower case is: GEEKS HELP

- **charAt()**

  This method is used to get the character at the given index

  Example:

```
let name = 'geeks help';
console.log(name);
console.log('Character at index 4 is '+ name.charAt(2));
```

output:

geeks help

Character at index 4 is e

- **includes()**

  It will return true if a string have given character or string otherwise it will return false

  Example:

```
let name = 'geeks help';
console.log(name);
console.log('Does string a includes r '+ name.includes('r'));
```

output:

geeks help

Does string a includes r false

- **concat()**

  It is used to concatenate two string.

  Example:

```
let name = 'raju webdev'
let role = ' Frontend Developer'
```

```
let result = name.concat(role);
console.log('Concatination of string ', name, ' and ', role, ' is
', result);
```

output:

Concatination of string raju webdev and fronend developer is raju webdev fronend developer

- **endsWith()**

  It return true is string ends with given character either it will return false

  Example:

```
let name = 'raju webdev'
console.log('Does string name ends with v', name.endsWith('v'));
```

output:

Does string name ends with v true

- **replace()**

  It is used to replace the given string with another one.

  Example:

```
let name = 'raju Web'
console.log(name.replace('Web', 'webdev'));
```

output:

raju webdev

- **split()**

  This method splits a string into an array of substrings.

  Example:

```
let name = 'raju Web'
console.log(name.split(' '));
```

output:

['raju', 'Web']

- **slice()**

  This will print the string between given indexes.

  Example:

```
let name = 'raju Web'
console.log(name.slice(0, 6));
```

  output:

  raju W

- **indexOf()**

  It will return the index of given character from a string.

  Example:

```
let name = 'raju Web'
console.log('The index of a in string name is: '+
name.indexOf('a'));
```

  output:

  The index of a in string name is: 1

Type conversion

- It is the process of converting one data type into another data type.
- Converting number into string is a type conversion.

Using type conversion.

- **1.**

```
let num = 2004;
console.log(String(num));
```

  output:

  2004

- **2.**

```
let booleanConversion = String(true);
console.log(booleanConversion)
```

output:

true

- **3.**

```
let myArray = [20, 4, 15, 22]
console.log(String(myArray))
```

output:

20,4,15,22

- **4.**

```
let str = Number(2004);
console.log(str);
```

output:

2004

- **5.**

```
let newNum = 2004;
console.log(newNum.toString());
```

output:

2004

Type coercion

- The type coercion is implicit whereas type conversion can be either implicit or explicit.

Using type coercion:

- **1.**

```
let a = 2004;
let b = '2004';
if(a===b){
  console.log('a is equal to b');
}
else{
  console.log('a is not equal to b');
}
```

output:

a is not equal to b

Boolean

- It return the value true or false.
- True means the value is true and false means value is false that means not correct.
- **false contains:** 0, -0, Null, NaN, undefined.

Using boolan:

- **1.**

```
let val1 = 'Yes';
let val2 = 'Yes';
console.log(val1 === val2);
```

output:

true

Symbols

- Symbols are nothing but similar to string.
- Symbols return a unique or new value.

Using Symbols:

- **1.**

```
let one = Symbol('Raju')
let two = Symbol('Raju')
console.log(one == two);
```

output:

false

Operators

- Operators are symbols used to perform some operations.
- Operations can be performed on one or more than one operands.

Types of operators:

- **Unary operators:**

  These operators are used on single variables.

  Unary operators are used to increment and decrement the value of variable.

  Some unary operators are: ++, --, ~!

  Example:

```
let num = 20;
num++;
console.log(num);
```

output:

2

- **Logical Operators:**

  Logical operators are used on one or more than one variables.

  Logical operators returns true or false value.

  Logical operators are: &&, ||, !

  Example:

```
let num = 20;
if(num>15 && num < 25){
```

```
    console.log('You are between 15 and 25'); }
```

output:

You are between 15 and 25

- **Relational Operators:**

  Relational operators describe the relation between two variables.

  These operators returns true or false.

  Relational operators are: ==, <, <=, >, >=, !=

  Example:

```
let num = 20;
if(num === 20){
    console.log('Num is 20');
}
else{
    console.log('Num is not equal to 20');
}
```

output:

Num is 20

- **Arithmetic Operators:**

  Arithmetic operators are used to perform arithmetic operations. Like addition, subtraction, multiplication, division, etc.

  Arithmetic operators are: +, -, *, /, %

  Example:

```
let num1 = 20;
let num2 = 4;
let result = 20+ 4
console.log(result);
```

output:

24

- **Assignment Operators:**

  Assignment operators are sued to assign value to the variables.

Assign the value by performing different operations.

Assignment operators are: =, +=, -=, *=, /=, %=, etc.

Example:

```
let num = 20;
num+=4;
console.log(num);
```

output:

24

Arrays

- Similar to a variable which used to store a collection of multiple values in a variable.
- Array can store primitive and Non-primitive(reference) data types.
- Always array starts from index 0 and last index is number of elements – 1.

Using Array:

- **Syntax of an Array:**

  const array_name = [item1, item2, item3,....item_n];

Example-1:

```
let myArray = [20,4, 15, 'Raj'];
console.log(myArray);
```

output: s

[20, 4, 15, 'Raj']

Example-2:

```
let secondArray = new Array(22, 224, 15, 'SkyBlue');
console.log(secondArray);
```

output:

[22, 224, 15, 'SkyBlue']

Array methdos:

- **length**

  It is used to get the length of an array

  Example:

```
let myArray = [20, 4, 15, 'Raju']
console.log(myArray.length);
```

  output:

  4

- **indexOf()**

  It is used to get the index of given character

  Example:

```
let myArray = [20, 4, 15, 'Raju']
console.log(myArray.indexOf(4));
```

  output:

  1

- **isArray()**

  It is used to get that does an object is an array or not if object is array then return true otherwise. False.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
let result = Array.isArray(fruits);
console.log(result);
```

  output:

  true

- **push()**

  It is used to put the element at the last index of an array.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
myArray.push("JavaScript Doctory");
console.log(myArray);
```

output:

[20, 4, 15, 'Raju', 'JavaScript Doctory']

- **unshif ()**

  It is used to put the element at the first index of an array.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
myArray.unshift('Geeks Help')
console.log(myArray);
```

  output:

  ['Geeks Help', 20, 4, 15, 'Raju']

- **pop ()**

  It is used to remove the last index element from an array.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
myArray.pop()
console.log(myArray);
```

  output:

  [20, 4, 15]

- **shift()**

  It is used to remove the first index element from an array.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
myArray.shift()
console.log(myArray);
```

  output:

  [4, 15, 'Raju']

- **reverse()**

  It will reverse the given array.

  Example:

```
const myArray = [20, 4, 15, 'Raju'];
console.log(myArray.reverse());
```

output:

['Raju', 15, 4, 20]

- **concat()**

  It is used to concatenate two arrays.

  Example:

```
const firstArray = [20, 4, 15, 'Raju'];
const secondArray = [23, 74, 34, 13, 15];
let newArray = firstArray.concat(secondArray)
console.log(newArray)
```

output:

```
[
  20,    4,  15,
  'Raju', 23, 74,
  34,    13, 15
]
```

Object

- Used to store multiple collection of data.
- It is non-primitive data types in JavaScript.
- Each member of an object is a key:value pair.

Object Declaration:

- **Syntax:**

```
let myObject = {
    key1: value1,
    key2: value2,
    key3: value3
}
console.log(myObject);
```

Example-1(Printing Object):

```
let obj = {
    name: 'Raju',
    page: 'raju_webdev',
    role: 'web developer'
}
console.log(obj);
```

output:

{ name: 'Raju', page: 'raju_webdev', role: 'web developer' }

Example-2(Accessing Object using for in loop):

```
let obj = {
    name: 'Raju',
    page: 'raju_webdev',
    role: 'web developer'
}
for(let key in obj){
    console.log(`${key}: ${obj[key]}`);
}
```

output:

name: Raju

page: raju_webdev

role: web developer

Example-3(Accessing name key in Object):

```
let obj = {
    name: 'Raju',
    page: 'raju_webdev',
    role: 'web developer'
}
console.log(obj.name);
```

output:

Raju

Example-4(Accessing page key in object):

```
let obj = {
    name: 'Raju',
    page: 'raju_webdev',
    role: 'web developer'
}
console.log(obj['page']);
```

output:

raju_webdev

Conditional Statements

- Use to control the flow of execution of the program.
- Conditional statements are similar to Decision-Making in real-life.
- The code is executed on the given return true or false.
- Conditional Statements are: if, if-else, nested if-else, if-else ladder, switch.

Conditional Statements are:

- **If statement:** In if statement, if the given condition is true then it will execute the code.

  Syntax:

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

Example:

```
let age = 20;
if(age>=18){
    console.log("You can vote to a party");
}
```

output:

You can vote to a party

- **If-else statement:** In the In if-else statement if the given condition is true then it will execute the code of if block otherwise the else block will be executed.

  Syntax:

```
if(condition){
    // if block code
}
else{
    // else block code
}
```

Example:

```
let age = 6;
if(age>18){
    console.log("You can go to the party");
}
else{
    console.log('You are under 18 you cannot go to the party.');
}
```

output:

You are under 18 you cannot go to the party.

- **Nested If-else statement:** In this if else statement there will be another if condition inside the main if condition

Syntax:

```
if(condition1){
    if(condition2){
        //first if code to be executed
    }
}
else{
    //else code to be executed
}
```

Example:

```
let age = 17;
if(age>15){
    if(age<60){
        console.log("You can go to the party");
    }
}
else{
    console.log('You are under 18 you cannot go to the party.');
}
```

output:

You can go to the party

- **If-else ladder statement:** In this statement, there is multiple if blocks and also known as else if ladder.

Syntax:

```
if(condition1){
    // condtion1 if code
}
else if(condition2){
    // condtion2 if code
}
```

```
else if(condition3){
    // condtion3 if code
}
else{
    // else code to be executed
}
```

Example:

```
let age = prompt('Enter you age: ')
if(age>35){
    console.log('You are greater than 35');
}
else if(age>25){
    console.log('You are greater than 25');
}
else if(age>15){
    console.log('You are greater than 15');
}
else{
    console.log('Please enter a valid age number');
}
```

Note : Use this code in your console of browser to show the output

---

Loops

- Loops are used to iterate the program on given conditions.
- OR loops are use to repeat the code block on given conditions until the condition false.
- Conditions in looping returns true or false.
- **Loops in JavaScript are:** for, while, do-while, for each.

Loops are:

- **for loop:** for loops are commonly used to run code until the given conditions false.

Syntax:

```
for(initialization; condition; increment/decrement){
    //code to be executed
}
```

Example:

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

output:

0

1

2

3

4

5

6

7

8

9

- **while loop:** In while loop the code will be executed until the given condition not become false.

Syntax:

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

Example:

```
let num = 0;
while (num<10) {
    console.log(num);
    num++;
}
```

output:

0

1

2

3

4

5

6

7

8

9

- **do-while loop:** This is similar to while loop but in this the code will be execute at least once then it will execute according to the condition.

  Syntax:

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

Example:

```
let num = 0;
```

```
do{
    console.log(num);
    num++;
}while(num<10)
```

output:

0

1

2

3

4

5

6

7

8

9

Break & Continue

When to use:

- Break and continue are used to jump the flow of execution from one place to another.
- Mostly used inside the control statements.
- These are used to exit a program from one control statement when the certain conditions meet.

Break: Break statement is used to exit the current block in a given program.

Example:

```
let num = 1;
do{
    console.log(num);
    if(num===4){
        break;
    }
    num++;
}while(num<=10);
```

output:

1

2

3

4

Continue: Continue statement is used to skip the current iteration and continue the next iteration.

Example:

```
for (let i = 0; i < 10; i++) {
    if(i===4){
        continue;
    }
   console.log("The number is " + i);
  }
```

output:

The number is 0

The number is 1

The number is 2

The number is 3

The number is 5

The number is 6

The number is 7

The number is 8

The number is 9

Function

- A function is a block of code that performs a specific task.
- A function is declared using the function keyword.
- The body of function is written within {}
- **In javascript functions are:** Regular function, Arrow function, Function Returning Something.

Functions are:

- **Regular function:** Regular functions created using function declaration or expressions.
  Arguments objects are available in regular functions.
  Syntax:

```
function function_name(){

    //code to be executed
}
```

Example:

```
function helloWorld() {
    console.log('Hello World to regular function');
}
helloWorld();
```

output:

Hello World to regular function

- **Arrow function:** Arrow functions is a new feature introduced in ES6, it enable writing concise functions in javascript.
  Single parameter function doesn't need paranthesis.
  Syntax:

```
const arrowFunction_name = ()=>{
    // code to be executed
}
```

Example:

```
const myArrowFunction = ()=>{
    console.log('This is arrow function in javascript');
}
greeting();
```

output:

This is arrow function in javascript

- **Function returns something:** A function which returns some value and store it into a variable.
  One line function doesn't need braces to return.
  Example:

```
const functionName = ()=>{
    return 'I am developer'
}

let myFunction = functionName();
console.log(myFunction);
```

output:

This is arrow function in javascript

Document Object Model (DOM)

- The DOM represents the whole HTML document and it is the structured representation of HTML document.
- With the help of DOM, we can interact with our web page.

Using DOM:

Syntax:

```
window.document;
```

Properties of DOM:

- **window.document**

  It is used to access complete document of any web page.

  Example:

```
let windowDocument = window.document;
console.log(windowDocument);
```

  output:

  Note: output will be in your console.

- **document.all**

  It will give the collection of all HTML tags in DOM.

  Example:

```
let documentAll =document.all;
console.log(documentAll);
```

  output:

  Note: output will be in your console.

- **document.all**

  It will give the body tag of our web page.

  Example:

```
let documentBody =document.body;
console.log(documentBody);
```

  output:

  Note: output will be in your console.

- **document.forms**

  It will give the collection of all forms available in our DOM.

  Example:

```
let documentForms=document.forms;
console.log(documentForms);
```

output:

Note: output will be in your console.

- **document.URL**

  The URL property is used to get the full URL of the documents.

  Example:

```
let documentURL=document.URL;
console.log(documentURL);
```

output:

Note: output will be in your console.

- **document.bgColor**

  This property is used to get the body color.

  Example:

```
document.bgColor = 'red';
console.log(document.bgColor);
```

output:

red

- **document.title**

  This property is used to get the title of the document.

  Example:

```
let documentTitle=document.title;
console.log(documentTitle);
```

output:

Note: output will be in your console.

- **window. alert**

  It is used to show an alert on the user window screen.

  Example:

```
window.alert('Welcome to javascript Doctory Series');
```

output:

Note: output will be in the alert box on the browser.

- **document. location**

  It is used to get the location of the document.

  Example:

```
let location = window.location;
console.log(location);
```

output:

Note: output will be in your console.

Element Selector

- Element selectors are used to select the HTML elements within a document using javascript.

Types of Element Selectors:

- There is mainly two types of element selectors in javascript.

  i). Single Element Selectors: This type of element selectors are used to select single tag or element from the document.

  ii). Multi Element Selector: This type of element selectors are used to select multiple tags or elements from the document.

Single Element Selectors:

- **getElementById();**

  This element selector is used to select the element by given id.

  Example:

```
let id = document.getElementById('myId');
```

```
console.log(id);
```

output:

Note: output will be in your console.

- **querySelector();**

  This element selector is used to select the element by given class or id. This element selector returns first match.

  Example:

```
let myPara = document.querySelector('p');
console.log(myPara);
```

  output:

  Note: output will be in your console.

- **ClassName ();**

  This selector is used to get and set the class to the element.

  Example:

```
let myPara = document.getElementById('SuperLogo');
myPara = myPara.className;
console.log(myPara);
```

  output:

  Note: output will be in your console.

- **className ();**

  This selector is used to get and set the class to the element.

  Example:

```
let myPara = document.getElementById('SuperLogo');
myPara = myPara.className;
console.log(myPara);
```

  output:

  Note: output will be in your console.

- **parentNode ();**

  This selector is used to get the parent of the selected element.

  Example:

```
let myPara = document.getElementById('childPara');
let parent = myPara.parentNode;
console.log(myPara);
console.log(parent);
```

output:

Note: output will be in your console.

Multi-Element Selectors:

- **querySelectorAll ();**

  This selector will return a list of the document's elements that match the specified group of selectors.

  Example:

```
let myPara = document.querySelectorAll('p');
console.log(myPara);
```

  output:

  Note: output will be in your console.

- **getElementByClassName ();**

  This element selector is used to select the collection of elements by their class name.

  Example:

```
let myParaClass = document.getElementsByClassName('paraClass');
console.log(myParaClass);
```

  output:

  Note: output will be in your console.

- **getElementByTagName ();**

  This element selector is used to get the collection of given tag name

  Example:

```
let myPara = document.getElementsByTagName('p');
console.log(myPara);
```

  output:

Note: output will be in your console.

Traversing DOM

- The nodes in the DOM are referred to as parents, children and siblings, depending on their relation to other nodes.

Properties of traversing the DOM:

- **parentNode:** It will get the parent node.
- **parentElement:** It will get the parent element node.
- **childNodes:** It is used to return a NodeList of child nodes.
- **children:** Used to get all the children on the selected element.

ParentNode:

The parent of any node is the node that is closer to the document in the DOM hierarchy. The parentNode property is read-only

- **parentNode**

  This element selector is used to select the element by given id.

  Example:

```
let myContainer = document.querySelector('.container');
mycontainer = myContainer.parentNode;
console.log(myContainer);
```

  output:

  Note: output will be in your console.

- **nextSibling**

  It will return the next node on the same tree level.

  Example:

```
let myContainer = document.querySelector('.container');
mycontainer = myContainer.nextSibling;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **nextElementSibling**

  It will return the next sibling element of the same tree level.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.nextElementSibling;
console.log(myContainer);
```

output:

Note: output will be in your console.

Child Nodes:

Child Nodes property is used to return a Nodelist of child nodes. ChildNodes includes text and comments.

- **childNodes**

  It taks text and comments of the DOM.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.childNodes;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **firstChild**

  It will give the first child node.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.firstChild;
```

```
console.log(myContainer);
```

output:

Note: output will be in your console.

- **lastChild**

  It will give the last child node.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.lastChild;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **nodeName**

  It will return the name of the current node as a string.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.nodeName;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **children**

  It is used to get all the children on the selected element.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.children;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **firstElementChild**

It will return the first child element of the selected element.

Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.firstElementChild;
console.log(myContainer);
```

output:

Note: output will be in your console.

- **lastElementChild**

  It will return the last child element of the selected element.

  Example:

```
let myContainer = document.querySelector('.container');
myContainer = myContainer.lastElementChild;
console.log(myContainer);
```

output:

Note: output will be in your console.

Modify DOM Elements:

- These methods are used to modify the selected elements on the DOM using some methods. Like creating some elements on DOM. And set or change the properties of the selected elements on the DOM.

Some Modifying Methods are

- **createElement();**

  This method is used to create element.

  Example:

```
let newList = document.createElement('li');
console.log(newList);
```

output:

Note: output will be in your console.

- **createTextNode ();**

  This method is used to create a text node for text.

  Example:

```
let text = document.createTextNode('This text is created text using createTextNode');
console.log(text);
```

  output:

  'This text is created text using createTextNode'

  Note: output will be in your console.

- **appendChild();**

  This method is used to append any element with respect to the selected element.

  Example:

```
let newList = document.createElement('li');
let text = document.createTextNode('This text is created text using createTextNode');
newList.appendChild(text)
console.log(newList);
```

  output:

  This text is created text using createTextNode

  Note: output will be in your console.

- **className();**

  This property is used to set class on the selected element.

  Example:

```
let newList = document.createElement('li');
newList.className = 'createdList'
console.log(newList);
```

  output:

  <li class='createdList'>

Note: output will be in your console.

- **id();**

  This property is used to set id on the selected element.

  Example:

```
let newList = document.createElement('li');
newList.id = 'myListId'
console.log(newList);
```

  output:

  <li id='myListId'>

  Note: output will be in your console.

- **replaceWith();**

  This method is used to replace a element with another element.

  Example:

```
let list = document.getElementById('oldList');

let newList = document.createElement('li')
list.replaceWith(newList)
```

  output:

  Note: output will be in your console/webpage.

- **removeChild();**

  This method is used to remove sub-element from the parent element.

  Example:

```
const mainList = document.querySelector('.list');

const removedElement = document.getElementById ('list-3')
mainList.removeChild(removedElement)
```

  output:

  Note: output will be in your console/webpage.

- **getAttribute();**

  This method is used to get the different properties of the selected element.

  Example:

```
const myList = document.getElementById ('list-3')
let listID = myList.getAttribute('id')
console.log('Id of list-3 is: ', listID);
```

  output:

  'Id of list-3 is: ', list-3

- **hasAttribute();**

  This method is used to know whether the selected element has or not the targeted property. It will return true or false

  Example:

```
const myList = document.getElementById ('list-3')
const hasAttribute = myList.hasAttribute('id')
console.log(hasAttribute);
```

  output:

  true

- **setAttribute();**

  This method is used to set the attribute on the selected element.

  Example:

```
const myList = document.getElementById ('list-3')
myList.setAttribute('class', 'list3Class');
```

- **removeAttribute();**

  This method is used to set the attribute on the selected element.

  Example:

```
const myList = document.getElementById ('list-3')
myList.removeAttribute('class');
```

- **innerText ();**

  This method is used to change the inner text of the selected element.

  Example:

```
const myList = document.getElementById ('list-3')
myList.innerText = 'This the changed element of the list'
```

Event Listeners:

- An event listener is a mechanism in javascript that tells that what event is occurred.
- Event listeners tell that what event has been occurred by the user on the DOM.
- In our JavaScript Doctory Series we will used event listeners by using addEventListner() methd.

Most common used Event Listeners are:

- ✓ onClick
- ✓ dblclick
- ✓ mouseover
- ✓ mousedown
- ✓ mouseup
- ✓ submit

- **click**

  This event listener used when a user click on an element.

  Example:

```
let submitBtn = document.getElementById('btn');

submitBtn.addEventListener('click', function(){

    console.log('Submit Button Clicked');

})
```

output:

Submit Button Clicked.

- **dbclick**

  This event listener used when a user double click on an element.

  Example:

```
let submitBtn = document.getElementById('btn');

submitBtn.addEventListener('dblclick', function(){

    console.log('Submit Button Double Clicked');

})
```

  output:

  Submit Button Double Clicked.

- **mouseover**

  This event is occur on the element when a mouse is used to move the cursor over the element.

  Example:

```
let submitBtn = document.getElementById('btn');

submitBtn.addEventListener('mouseover', function(){

    console.log('Submit Button Mouse Over');

})
```

  output:

  Submit Button Mouse Over

- **mouseup**

  This event listener used when a user moveup the mouse on the element.

  Example:

```
let submitBtn = document.getElementById('btn');

submitBtn.addEventListener('mouseup', function(){

    console.log('Submit Button Mouse Up occurred.');

})
```

  output:

Submit Button Mouse Up occurred.

- **submit**

  It is used when we want to submit the specified form.

  Example:

```
let submitForm = document.getElementById('myForm');

submitForm.addEventListener('submit', function(e){

     e.preventDefault();

    console.log('Form submit button clicked.');

})
```

output:

From submit button clicked.

Math Object:

- in javascript, math object is used to perform the mathematical properties in our program.
- Math object is static and it has no constructor.

Most used Math object properties:

- ✓ PI
- ✓ ceil
- ✓ floor
- ✓ round
- ✓ random
- ✓ sqrt
- ✓ min, max, etc.

Using Math properties:

- **PI**

  It is used to get the value of PI.

Example:

```
let PIvalue = Math.PI;
console.log('Value of PI is: ', PIvalue);
```

output:

Value of PI is: 3.141592653589793

- **ceil**

  It is used to get the higher value.

  Example:

```
let higherValue = Math.ceil(5.2);

console.log('Higher value of 5.2 is: ', higherValue);
```

  output:

  Higher value of 5.2 is: 6

- **floor**

  It is used to get the least value.

  Example:

```
let floorValue = Math.floor(5.4);
console.log('Floor value of 5.4 is: ', floorValue);
```

  output:

  Higher value of 5.2 is: 5

- **round**

  It returns a rounded number to the nearest interger.

  Example:

```
let myNum = Math.round(2.4);
console.log('Nearest value to 2.4 is: ', myNum);
```

  output:

  Nearest value to 2.4 is: 2

- **abs**

  It is used to get the absolute value of the given number. It changes –ve to +ve.

Example:

```
let absValue = Math.abs(-20);
console.log('absValue of -20 is: ', absValue);
```

output:

absValue of -20 is: 20

- **sqrt**

  It is used to get the square root of the given number.

  Example:

```
let sqrNum = Math.sqrt(36);
console.log('Square of 36 is: ', sqrNum);
```

output:

Square of 36 is: 6

- **min**

  It is used to get the minimum value from the different values.

  Example:

```
let minValue = Math.min(24, 43, 4, 54, 343, 54, 32);
console.log('Minimum value is: ', minValue);
```

output:

Minimum value is: 4

- **max**

  It is used to get the maximum value from the different values.

  Example:

```
let maxValue = Math.max(24, 43, 4, 54, 343, 54, 32);
console.log('Maximum value is: ', maxValue);
```

output:

Maximum value is: 343

- **pow**

  It is used to get the power of the given number. And it take value and power.

Example:

```
let power = Math.pow(2,3);
console.log('3 times power of 2 is: ', power);
```

output:

3 times power of 2 is: 8

- **E**

  Math.E is used to get the Euler's number.

  Example:

```
let eulerNum = Math.E;
console.log("Euler's Number is: ", eulerNum);
```

output:

Euler's Number is: 2.718281828459045

- **random**

  Random is the most used property of Math object. This property is used to get random numbers.

  Example:

```
let randomNumber = Math.random();
console.log('Random Number is: ', randomNumber);
```

Note: Output will be different every time

Date and Time:

- Most important concept throw javascript learning these concepts are used to get the current Date and Time and also used to perform many other operations on the website.

Date and Time properties are:

- ✓ new Date();
- ✓ getDate();
- ✓ getHours();
- ✓ getMinutes();

- ✓ getSeconds();
- ✓ getTime();, etc.

Using Date and Time properties:

- **new Date();**

  Used to get the current Time and Date.

  Example:

```
let today = new Date();
console.log(today);
```

  Note: It will print the current date on your console.

- **getDate();**

  Used to get the date.

  Example:

```
let today = new Date();
console.log(today.getDate());
```

- **getHours ();**

  Used to get the current hour.

  Example:

```
let today = new Date();
console.log(today.getDate());
```

- **getMinutes();**

  Used to get the minutes.

  Example:

```
let todayMinutes = new Date();
console.log(todayMinutes.getMinutes());
```

- **getSeconds();**

  Used to get the seconds.

  Example:

```
let todaySeconds = new Date();
console.log(todaySeconds.getSeconds());
```

- **getMonth();**

  Used to get the month.

  Example:

```
let thisMonth = new Date();
console.log(thisMonth.getMonth());
```

- **getFullYear();**

  Used to get the full year.

  Example:

```
let thisYear = new Date();
console.log(thisYear.getFullYear());
```

- **setDate();**

  Used to set the date.

  Example:

```
let date  = new Date();
date.setDate(20);
console.log(date);
```

- **setFullYear();**

  Used to set the full year.

  Example:

```
let year  = new Date();
year.setFullYear(2023);
```

```
console.log(year);
```

- **setHours();**

  Used to set the hours.

  Example:

```
let hours = new Date();
hours.setHours(14);
console.log(hours);
```

- **setMinutes();**

  Used to set the minutes.

  Example:

```
let minutes = new Date();
minutes.setMinutes(24);
console.log(minutes);
```

- **setSeconds();**

  Used to set the seconds.

  Example:

```
let seconds = new Date();
seconds.setSeconds(24);
console.log(seconds);
```

- **setMonth();**

  Used to set the month.

  Example:

```
let month = new Date();
month.setMonth(12);
console.log(month);
```

Local & Session Storage:

Local Storage:

- In local storage, the data is store in the local memory of the web browser and local storage data don't have any expiry date. And the data is stored in key value pairs.

Local storage Methods:

- ✓ setItem();
- ✓ getItem();
- ✓ removeItem();
- ✓ clear();

- **setItem();**

  It is used to set the items in the browser storage.

  Example:

```
let myData = localStorage.setItem('Name', 'Raju');
```

  It will store 'Name' as key and 'Raju' as a value in local storage of the web browser.

- **getItem();**

  It is used to get the items from the browser storage.

  Example:

```
let myData = localStorage.setItem('Name', 'Raju');
let getData = localStorage.getItem('Name');
console.log(getData);
```

  Output: Raju

- **removeItem();**

  It is used to remove the item from the browser storage.

  Example:

```
localStorage.removeItem('myData');
```

- **clear();**

  used to clear the browser local storage.

Example:

```
localStorage.clear();
```

Session Storage:

- In session storage, the data is stored for the particular session. And the data will be lost when the web browser is closed.

Session Storage Methods:

- ✓ setItem();
- ✓ getItem();
- ✓ removeItem();
- ✓ clear();

- **setItem();**

  Example:

```
let myData = sessionStorage.setItem('Name', 'Raju');
```

- **getItem();**

  Example:

```
let myData = sessionStorage.setItem('Name', 'Raju');
let getData = sessionStorage.getItem('Name');
console.log(getData);
```

  Output: Raju

- **removeItem();**

  Example:

```
sessionStorage.removeItem('myData');
```

- **clear();**

  Example:

```
sessionStorage.clear();
```

Class:

- Classes are a template for creating objects.
- It is just like a blueprint template which is used by objects to perform different operations.
- Classes can be only logically represented in any programming language.
- We can get the value of different properties of class using . (dot) with object.

Object:

- An object is an element (or instance) of a class. Objects have the behaviors of their class.

Constructor:

- Constructor is like a function which will be run after the creation of the object.

Creating Class and different Objects:

Example:

```
// Creating student class
class student{
    constructor(getName, getClass, getRoll, getSection){
        this.name = getName;
        this.class = getClass;
        this.roll = getRoll;
        this.section =getSection;
    }
}
// Creating Objects for student1 class
let student1 = new student('Raju', 'BCA', 4, 'A');
console.log(student1);

// Creating Objects for student2 class
let student2 = new student('Jassi', 'BA', 13, 'C');
console.log(student2);
```

output:

student {name: 'Raju', class: 'BCA', roll: 4, section: 'A'}
student {name: 'Jassi', class: 'BA', roll: 13, section: 'C'}

Inheritance:

- Inheritance means get the some properties of already created class to create new class. And **extends** keyword is used to create inheritance.

Super in Inheritance:

- super keyword is used to get the constructor of the parent class or the super class.

Creating Inheritance:

Example:

```
// Creating student class
class student{
    constructor(getName, getClass, getRoll, getSection){
        this.name = getName;
        this.class = getClass;
        this.roll = getRoll;
        this.section =getSection;
    }
}
// Inheritance
class developer extends student{
    constructor(getName, getClass, getLanguage, getRole){
        super(getName, getClass);
        this.language = getLanguage;
        this.role = getRole;
    }
}

// Creating Objects for student1 class
let student1 = new student('Jassi', 'BCA', 4, 'A');
console.log(student1);
```

```
// Creating Inheritance Object
let newDeveloper = new student('Raj', 'BA', 'JavaScript', 'Web
Developer');
console.log(newDeveloper);
```

output:

student {name: 'Jassi', class: 'BCA', roll: 4, section: 'A'}

student {name: 'Raj', class: 'BA', role: 'JavaScript', section: 'Web Developer'}

Synchronous and Asynchronous:

Synchronous

- In synchronous programming the things will happen one at a time. More than one thing cannot happen at one time.
- In this statement of the code gets executed one by one

Example:

```
console.log('Raju Webdev');

console.log('Geeks Help');

console.log('Tell me about yourself on website geekshelp');
```

Output:

Raju Webdev

Geeks Help

Tell me about yourself on website geekshelp

Asynchronous

- In Asynchronous programming code allows the program to executed in the background and another program will be executed that time for the further execution.
- In this statement more than one program are executed at a time.

Example:

```
console.log('Raju Webdev');


setTimeout(() => {
    console.log('Hello Web Developers');
}, 2000)
```

Output:

Raju Webdev

Hello Web Developers //it will print on console after two seconds

Synchronous	Asynchronous
Operations task are performed one at a time.	Operations will execute in the background of another operation.
Another operation or code will executed after execution of previous code.	Multiple operations can be executed at one time.
It waits for each operation to complete.	It never waits for each operation to complete.

Callback:

- We can pass a function as an argument to a function. The function that is passes as an argument inside of another function is called a callback function.
- The function which is executed after the execution of another function that is finished.

Why callback function?

- The callbacks are used to make sure that a function is not going to execute before a task is completed but will execute right after the task has completed.
- We use callback functions are used to make the asynchronous programming.

Syntax:

```
// function
function function_name(firstArgs, callback){
    // code to be executed
    callback();
}

// callback function
function callback_function_name(){
    //code to be executed
}
// passing function as an argument
function_name('firstArgs', callbackFunction);
```

Example:

```
// function
function greet(getName, callback){
    console.log('Hi ', getName);
    callback();
}

// callback function
function callMe(){
    console.log('I am callback function');
}
// passing function as an argument
greet('Raju',callMe);
```

Output:

Hi Raju

I am callback function

Async / Await:

Async:

- Async keyword is used with a function to represent that the function is an asynchronous function. The async function returns a promise.

Syntax:

```
async function_name(param1, param2, ...paramN){
    // code to be executed
}
greet('Raju',callMe);
```

Example:

```
async function greet(){
    return 'Good Morning Developers'
}
let myName = greet();
console.log(myName);
```

Output:

Good Morning Developers //output will be as a promise

Await:

- The await keyword is used inside the async function to wait for the asynchronous operation.
- await pauses the async function until the promise returns a result.

Syntax:

```
let result = await Promise;
```

Example:

```
async function getUsersData() {
    console.log('Before getting Data');
    const response = await fetch('https://api.github.com/users');
    console.log('After Fetching Data');
    const githubUsers = await response.json();
    return githubUsers;
}

let printData = getUsersData();
printData.then((data) => {
    console.log(data);
})
```

Output:

Note: output will be in your console

Fetch API:

- Fetch API is an interface which allows us to make HTTP requests to servers from the user web browsers.
- Fetch is based on async and await programming nature.
- fetch api uses two (.then) first to resolve the response from the server and second to get the data from the response.

Syntax:

```
fetch(url).then((response)=>{
    // response code
}).then((data)=>{
    // data
})
```

readme.txt:

```
Hello Developers this is our readme file of JavaScript Doctory Series
```

Example:

```
function readFile(){
    fetch('readme.txt').then((response)=>{
        return response.text();
    }).then((data)=>{
        console.log(data);
    })
}
readFile();
```

Output:

Hello Developers this is our readme file of JavaScript Doctory Series

---

Error Handling:

- Error handling is used to stop the generated error message and show a normal message instead of showing error.
- In javascript errors are handled at the runtime of the program.
- In javascript try and catch block are used to handle errors. And one more block is finally is also used to handle errors.

Syntax:

```
try{
    // try block code
}
catch(error){
    // catch block code
}
```

```
finally{
    // finally block code
}
```

Try Catch:

- In try catch block if there is any error then the catch block will show the message instead of showing any error.

Example:

```
try{
    console.log('Try Block executed');
    name();
}
catch(error){
    console.log('Catch block executed');
}
```

Output:

Try Block executed

Catch Block executed

Try Catch Finally:

- In try catch and finally block the try and catch block will execute as we discuss in previous example but the finally will show the message every time.

Example:

```
try{
    console.log('Try Block executed');
    name();
}
catch(error){
    console.log('Catch block executed');
}
finally{
    console.log('Finally block executed');
}
```

Output:

Try Block executed
Catch Block executed
Finally Block executed

Promises:

- Promises is an object and promises are just like a promise which we do in our real-life.
- Promise performs different actions in which .then() used for promise is successfully fulfilled or resolved and .catch() for reject or an error.
- The function which is executed after the execution of another function that is finished.

Syntax:

- To create a promise, we use the Promise() constructor.

```
let myPromise = new Promise(function(resolve, reject){
    // code to be executed
});
```

Example:

```
let web = new Promise(function(resolve, reject){
    const first = 'geekshelp';
    const second = 'geekshelp';
    if(first===second){
        resolve();
    }
    else{
        reject();
    }
});

web.then(function(){
    console.log('Your promise is resolved.');
}).catch(function(){
    console.log('Your promise is rejected.');
})
```

Output:

Your promise is resolved.

Cookies:

- Cookies are some data & text files which stored on the webpage.
- A web browser stores this information at the time of browsing.
- Basically, it stores the information as a string in the form of a name-value pair.

Why cookies are used in a website?

- To store the user's information in the web page.
- For remember the data of the used for visiting website next time.
- Help a user to give the recommendation according to their previous activities.

Why cookies are used in a website?

```
document.cookie = "name=value";
```

Example:

```
document.cookie = "website=geekhelp";
```

Dates in Cookies:

- You can also store the date in user's web page. And we can also store multiple information by sepration semicolon(;).

Multiple cookies:

```
document.cookie = "username = raju; website=geekshelp";
```

Reading Cookies:

```
let myCookie = document.cookie = "username = raju; website = geekshelp";
console.log(myCookie);
```

Output:

username = raju; website = geekshelp

Update Cookies:

As we create the cookies insame way we can update the cookies by changing value.

```
let myCookie = document.cookie = "username = raju; website = geekshelp";
console.log(myCookie);

myCookie = document.cookie = "username = raju_webdev; website = geekshelp ";
console.log(myCookie);
```

Output:

username = raju; website = geekshelp

username = raju_webdev; website = geekshelp

Delete Cookie:

We can delete cookie by set the expires parameter to a past date.

```
document.cookie = "name = ; expires= Thu, 04 Aug 2022 00:00:00 IST; path=/;";
```

Destructuring:

- It allows us to extract data from arrays, objects and maps or set them into new distinct variable.
- It allows us to extract multiple properties and items from an array at a time.
- Destructuring used to break down the complex structure into small parts to easily understand.

Where Destructuring used?

- Array Destructuring
- Object Destructuring

Syntax:

```
let {var1, var2} = {var1:value1, var2:value2,...}
```

OR

```
let [var1, var2] = [value1, value2]
```

Array Destructuring:

- It means break down a complex structure into simpler parts.

Example:

```
const myArray = ['Orange', 'Apple', 'Banana'];

const [x,…z] = myArray;

console.log(myArray);

 })
```

Output:

['Orange', 'Apple', 'Banana']

Rest Operator

- You can put all the remaining elements of any array in a new array using rest operator(…)

Example:

```
const fruitsArray = ['Orange', 'Apple', 'Banana', 'Grapes', 'Watermelon',
'Kivi'];
const [x,y,...z] = fruitsArray;
console.log(fruitsArray);
```

Output:

['Orange', 'Apple', 'Banana', 'Grapes', 'Watermelon', 'Kivi',]

Object Destructuring:

- As array destructuring, object destructuring also break down a complex object into smaller parts to easily access.

Example:

```
const myData = {

    name: 'raju_webdev',
```

```
    age: 19,
    myClass: 'BCA',
    website: 'Geeks Help',
    role: 'Frontend Developer'
};

const {name, myClass, website} = myData;
console.log(name, myClass, website);
```

Output:

raju_webdev BCA Geeks Help

OR

Example:

```
const myData = {
    name: 'raju_webdev',
    age: 19,
    myClass: 'BCA',
    website: 'Geeks Help',
    role: 'Frontend Developer'
};

const {name,...a} = myData;
console.log(name, a);
```

Output:

raju_webdev {age: 19, myClass: 'BCA', website: 'Geeks Help', role: 'Frontend Developer' }

Map:

- In javascrit, map holds the key-value pairs. And it can be any type of key or value.
- In map the key can be any type of data type.
- The Map object can hold objects and primitive values as either key or value pairs.

Creating Map:

- We can create map in javascript by passing an array to the map.
- And another way is by creating Map and use Map.set()

Example:

```
const names = new Map([['raju', 2004],['geekshelp', 2021], ['webdev', 2022]]);

console.log(names);
```

Output:

Map(3) {'raju',=>2004, 'geekshelp'=>2021, 'webdev',2022 }

Map Methods:

- **new Map();**

  Creates a new Map object

  Example:

```
const names = new Map([['raju', 2004],['geekshelp', 2021],
['webdev', 2022]]);

console.log(names);
```

  Output:

  Map(3) {'raju',=>2004, 'geekshelp'=>2021, 'webdev',2022 }

- **set();**

  It will sets the value for a key in a Map.

  Example:

```
const myMap = new Map();
let value1 = 'Age';
let value2 = 'Year';

myMap.set(value1, '20');
myMap.set(value2, '3rd Year');
console.log(myMap);
```

Output:

Map(2) {Age',=> '20', 'Year'=>'3rd Year'}

- **get();**

  It is used to get the value for a key in a Map.

  Example:

```
const myMap = new Map();
let value1 = 'Age';
let value2 = 'Year';

myMap.set(value1, '20');
myMap.set(value2, '3rd Year');
console.log(myMap);

let val = myMap.get(value1);
console.log(val);
```

Output:

Map(2) {Age',=> '20', 'Year'=>'3rd Year'}

20

- **size**

  It will return the number of elements in Map.

  Example:

```
console.log('Size of myMap is: ', myMap.size);
```

Output:

Size of myMap is: 2

- **keys()**

  It is used to get the keys from a Map.

  Example:

```
const myMap = new Map();
let value1 = 'Age';
let value2 = 'Year';

myMap.set(value1, '20');
myMap.set(value2, '3rd Year');

for(let key of myMap.keys()){
    console.log('Key of myMap is: ', key);
}
```

  Output:

  Keys of myMap is: Age

  Keys of myMap is: Year

- **values()**

  It is used to get the value from a Map.

  Example:

```
const myMap = new Map();
let value1 = 'Age';
let value2 = 'Year';

myMap.set(value1, '20');
myMap.set(value2, '3rd Year');

for(let value of myMap.values()){
    console.log('Values of myMap is: ', value);
}
```

  Output:

  Values of myMap is: 20

  Values of myMap is: 3rd Year

- **has()**

  This method will return true if the given key exists in a Map.

  Example:

```
let mapAge = myMap.has('Age');
console.log(mapAge);
```

  Output:

  true

Set:

- Set in javascript is a collection of values.
- It is used to sto4re unique values. And the value can be of any data types.
- The value in set are unique, So each value can only occr once in a set.

Initilize an empty Set:

```
const names = new Map([['raju', 2004],['geekshelp', 2021], ['webdev', 2022]])
console.log(names);
```

Output:

Map(3) {'raju',=>2004, 'geekshelp'=>2021, 'webdev',2022 }

Creating Set:

- By passing an array to new Set();
- Create a new Set and use add() to add values and variables.

Set Methods:

- **new Set();**

  This will initialize an empty Set.

  Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
console.log(mySet);
```

  Output:

  Set(5) { 'raju', 'geekshelp', 20, 4 , 'web developer' }

- **add();**

  It is used to add new element to the set.

  Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
mySet.add('I am a developer');
console.log(mySet);
```

  Output:

  Set(6) { 'raju', 'geekshelp', 20, 4 , 'web developer', 'I am a developer' }

- **delete();**

  It will remove the specified element form the set.

  Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
mySet.delete('web developer');
console.log(mySet);
```

  Output:

  Set(5) { 'raju', 'geekshelp', 20, 4 }

- **size();**

  It will return the number of elements in set.

  Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
console.log(mySet.size);
```

  Output:

  5

- **has();**

  This method will return true if the given key exists in a set.

  Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
console.log(mySet.has('raju'));
```

  Output:

  True

- **values();**

  It will returns an object which contains all the values in a set.

Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
console.log(mySet.values());
```

Output:

[Set Iterator] { 'raju', 'geekshelp', 20, 4 , 'web developer' }

Iterating a Set

Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
for (let value of mySet) {
    console.log('Value of set is: ', value);
}
```

Output:

Value of set is: raju

Value of set is: geeks help

Value of set is: 20

Value of set is: 4

Value of set is: web developer

Iterating a Set using forEach

Example:

```
let mySet = new Set(['raju', 'geeks help', 20, 4, 'web developer']);
mySet.forEach((value)=>{
    console.log('Value of set is: ', value);
});
```

Output:

Value of set is: raju

Value of set is: geeks help

Value of set is: 20

Value of set is: 4

Value of set is: web developer

Array Map:

- It creates an array by calling a specific function on element of the array.
- Generally, map method is used to iterate over an array.
- Map method calls the function for every element of array.

Syntax:

```
array.map(function(value, index, array){// code to be executed});
```

Parameters of map:

- **functions(value, index, array):** It is a required parameter and it runs on each element of array.
- **value:** Required parameter & it holds the value of current element.
- **Index:** It is an optional parameter and it holds the index of the current element.
- **array:** It is optional parameter and it holds the array.

Iterating Array:

- **Using traditional mehod**

  Example:

```
let myArray = ['raju_webdev', 'geeks help', 'instagram', 'web devp'];

for (let i = 0; i < myArray.length; i++) {
    console.log(myArray[i]);
}
```

  Output:

  raju_webdev
  geeks help
  instagram
  web devp

- **Using map mehod**

  Example:

```
let myArray = ['raju_webdev', 'geeks help', 'instagram', 'web devp'];
```

```
myArray.map(function(value, index){
    console.log(value, 'at index', index);
})
```

Output:

raju_webdev at index 0

geeks help at index 1

instagram at index 2

web devp at index 3

- **Using map mehod with arrow function**

Example:

```
let myArray = ['raju_webdev', 'geeks help', 'instagram', 'web devp'];

myArray.map((value, index)=>{
    console.log(value, 'at index', index);
})
```

Output:

raju_webdev at index 0

geeks help at index 1

instagram at index 2

web devp at index 3

For...in and For...of

for...in

- for in allow us to iterate all the property key of an object.
- In object, for in loop is used to access the value of the key.
- It returns a key when we use it with object.

Syntax:

```
for(let key in object){
    // code to be executed
}
```

Example:

```
const myData = {
    name: 'Raju Webdev',
    language: 'JavaScript',
    class: 'BCA',
    website: 'geekshelp.in'
}

// Iterating myData using for in loop
for(let key in myData){
    console.log(myData[key]);
}
```

Output:

Raju Webdev

JavaScript

BCA

geekshelp.in

for...of

- for of loop allow us to iterate over the iterable object, arrays, sets, maps, etc.

Syntax:

```
for(let data of array){
    // code to be executed
}
```

Example:

```
const weekDays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'];

// Iteraitng weekDays array using for of loop
for(let today of weekDays){
    console.log(today);
}
```

Output:

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Sunday

Array Filter

- Array filter is used when we want to apply some condition on array.
- In the array, filter executes the function for each element of the array.
- It creates a new filtered array on the given conditions.

Syntax:

```
array.filter(function(element, index, array){
    // code
})
```

Parameters of filter:

- **functions(element, index, array):** It is a required parameter and it runs on each element of array.
- **value:** It holds the value of current element.
- **Index:** It holds the index of the current element.
- **array:** It is optional parameter and it holds the array.

Using Traditional Method:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

for (let i = 0; i < values.length; i++) {
    if (values[i] >= 19) {
        console.log(values[i]);
    }
}
```

Output:

20

61

420

22

204

19

Using filter Method:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

let newArray = values.filter(function(value){
    if(value<=19){
        return value
    }
})
console.log(newArray);
```

Output:

[4, 3, 1, 13, 15, 19]

Using filter with arrow function:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

let newArray = values.filter((value)=> value<=19);
console.log(newArray);
```

Output:

[4, 3, 1, 13, 15, 19]

Array Reduce:

- Array reduce is used to perform some operation on array to get a single value.
- array.reduce doesn't execute the function for the empty array element.
- It doesn't change the original array but it give a single value output.

Syntax:

```
array.reduce(function(previous, current, index){
    // code
}, initialValue)
```

Parameters of reduce:

- **functions(previous, current, index):** It is a required parameter and it runs on each element of array.
- **previous:** It holds the value of previous element.
- **current:** It holds the value of current element.
- **index:** It holds the index of the current element.

Using Traditional Method:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

let sum = 0;
for(let i=0; i<values.length; i++){
    sum += values[i];
};

console.log(sum);
```

Output:

782

Using reduce Method:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

let valuesSum = values.reduce(function(previous, current){
    return previous + current;
}, 0);

console.log(valuesSum);
```

Output:

782

Using reduce with arrow function:

```
const values = [20, 4, 3, 61, 1, 13, 420, 22, 15, 204, 19];

let valuesSum = values.reduce((previous, current)=>{
    return previous + current;
}, 0);

console.log(valuesSum);
```

Output:

782

Hoisting:

- Hoisting means moving the variable, function and class declaration to the top of their scope.
- Variables and class with declaration in javascript are hoisted.
- Functions with function keyword are by default hoisted in javascript.

Syntax:

```
array.reduce(function(previous, current, index){
    // code
}, initialValue)
```

Hoisting used on:

- Variable Hoisting
- Function Hoisting
- Class Hoisting

Variable Hoisting:

- It is the mechanism to move the declaration of the variable to the top of their scope.
- Variable created using var keyword will give undefined if we use it before initialization.

Example:

```
console.log(myName);
var myName= 'raj';
```

Output:

Undefined

- Variable defined with let and const are hoisted to the top of the block , but not initialized.

Example:

```
console.log(myName);
let myName= 'raj';
```

Output:

ReferenceError: Cannot access 'name' before initialization

```
console.log(myName);
const myName= 'raj';
```

Output:

ReferenceError: Cannot access 'name' before initialization

Function Hoisting:

- Function in javascript using function keyword are hoisted.

Example:

```
myName('raju_webdev');

function myName(name){
    console.log('My Name is: ', name);
}
```

Output:

My Name is: raju_webdev

Arrow function Hoisting:

- Arrow Function in javascript are not hoisted.

Example:

```
myName('raju_webdev');

const myName = (name)=>{
    console.log('My Name is: ', name);
}
```

Output:

ReferenceError: Cannot access 'name' before initialization

Class Hoisting:

- Class with declaration are hoisted.
- Any code of class which is used before it's initialization will throw an error

Example:

```
const st1 = new student('Raju', 'BCA');
console.log(st1.name);
console.log(st1.class);
class student{
    constructor(getName, getClass){
        this.name = getName;
```

```
        this.getClass = getClass;
    }
}
```

Output:

ReferenceError: Cannot access 'name' before initialization

---

setTimeout

- This method is used to call a function after a number of given specified time in milliseconds.
- It is used to perform asynchronous programming.
- It executes only once.
- We can also set parameters in setTimeout()

Syntax:

```
setTimeout(function, timeDelay, param1, param2, param3,..., paramN);
```

- **function:** A function which contains a block of code.
- **timeDelay:** Time which defines that after how many milliseconds function will be execute.
- **params:** These are optional parameters.

Example:

```
function printOutput(){
    setTimeout(function(givenName, givenRole){
        let name = givenName;
        let role = givenRole;
        console.log('I am ', name, 'a', role);

    }, 2000, 'Raju', 'Frontend Developer');
}

console.log('Starting...');
printOutput();
```

Output:

Starting...

I am Raju a Frontend Developer

---

setInterval

- This method is used to execute a function repeatedly within a given time interval.
- It takes function and timeDelay as parameter and arguments as optional.

Syntax:

```
setInterval(function, timeDelay,args1, args2, args3,..., argsN);
```

- **function:** Function which which will execute after the delay time.
- **timeDelay:** Time which defines that after how many milliseconds function will be execute.
- **args:** These are optional arguments.

Example:

```
setInterval(function(givenName, givenRole){

        let name = givenName;

        let role = givenRole;

        console.log('I am ', name, 'a', role);


    }, 2000, 'Raju', 'Frontend Developer');
```

Output:

I am Raju a Frontend Developer
I am Raju a Frontend Developer
I am Raju a Frontend Developer
.....

*** Don't Stop Learning || Keep Growing || Keep Learning ***

JavaScript Project Ideas

1) **Digital Clock:** This simple project you can create using Date.
2) **Calculator:** Most of the senior developers and mentor recomanded to create a simple calculator app so you can create a calculator as a beginner level project.
3) **Note App:** You can create a note app as a beginner and can store the notes in the local storage of the browser.
4) **Slider:** It is mostly created javascript project which you can create it will have the previous and next buttion to slide the different images.
5) **News Web:** Create a news website using fetch api to show the real time news.
6) **Form Validation:** Form validation is mostly recommended project for beginners. So as a beginner-level project you can create a project using javascript with the specified validations.
7) **Music Player:** Create a music player which have same given music to play**.**
8) **Speech-Recognition System:** Create a speech recognition system like Jarvis.
9) **Flappy Bird Game:** It is the most common game most of the people had play this game, So you can also create this game as your javascript project.
10) **Real-Time Chat Application:** You can also create a chat app using javascript and NodeJs as a server.
11) **Blog Web App:** It will give the functionality to the use to signup or login. And where user can like, comment, share, etc. the blogs.
12) **E-Commerce:** This project will be like Amazon and Flipcart.
13) **College Library:** Create a simple library using classes and create add book, issue book, delete book, etc. functionality to the web app.
14) **Hospital Management:** A website which will give all the details from doctor to patient. And and also will generate the reports.

Web Development Roadmap

This web develoment roadmap is a part of our JavaScript Doctory Series. And in this web development roadmap our main focus is to make you a Full Stack Web Developer using JavaScript as a main language.

- Using javascript you can go in MERN, MEAN and MEVN stack. You can choose according to your interest.

- **MERN:** It is the most popular and demanding technology. And is for the beginners. You can learn this by following the given steps:
  - ✓ **HTML:** HTML is used to create the structure of the web page.
  - ✓ **CSS:** CSS is used to implement the design to the web page.
  - ✓ **JavaScript:** JavaScript is used to add additional functionality to the web page.
  - ✓ **ReactJs:** JavaScript library to build single page web applications.
  - ✓ **NodeJs:** Server environment.
  - ✓ **Express:** It the popular framework for NodeJs.
  - ✓ **MongoDB:** NoSQL database to deal with data.

- **MEAN:** It is also the most popular and demanding technology. You can learn this by following the given steps:
  - ✓ **HTML:** HTML is used to create the structure of the web page.
  - ✓ **CSS:** CSS is used to implement the design to the web page.
  - ✓ **JavaScript:** JavaScript is used to add additional functionality to the web page.
  - ✓ **Angular:** It is very popular framework to build dynamic web applications.
  - ✓ **NodeJs:** Server environment.
  - ✓ **Express:** It the popular framework for NodeJs.
  - ✓ **MongoDB:** NoSQL database to deal with data.

- **Additional Skills:** Now after leaning these technologies build some projects which will help you to become job ready.

❖ **Additional Skills:**

- ✓ **Git:** It is a version control system which will help you to manage and keep track of your code history. And it is a software.
- ✓ **GitHub:** It is a hosting service and it is hosted on the web.
- ✓ **npm:** npm stands for Node Package Manager. As it's name suggest it is a package manager for nodeJs.

Learning Resources:

Sr. No.	Resources	Links
01.	HTML by w3schools	(Learn Now)
02.	CSS by w3schools	(Learn Now)
03.	JavaScript by w3schools	(Learn Now)
04.	Web Development Course by CodeWithHarry	(Learn Now)
05.	JavaScript by CodeWithHarry	(Learn Now)
06.	JavaScript by Nisha Singla	(Learn Now)
07	ReactJS by Nisha Singla	(Learn Now)
08	ReactJS by CodeWithHarry	(Learn Now)
09	ReactJs by Technical Suneja	(Learn Now)
10.	ReactJS by Thapa Technical	(Learn Now)
11.	NodeJS by CodeWithHarry	(Learn Now)
12.	Nodejs by w3schools	(Learn Now)
13.	Express by CodeWithHarry	(Learn Now)
14.	Express by Thapa Technical	(Learn Now)
15.	Angular by Nisha Singla	(Learn Now)
16.	Angular in one video by CodeWithHarry	(Learn Now)
17.	MongoBD by w3schools	(Learn Now)
18.	MongoBD by CodeWithHarry	(Learn Now)

‹Contact Us for any query and Guidance/›