

JavaScript Fundamentals

i. Introduction to Javascript

◇ Why do we need Javascript ?

It brings life to your HTML page. If you want to add any functionality on any HTML element on a web page, then we need JS.

◇ Why do we need Javascript ?

It is scripting language that enables you to create dynamically updating content, control multimedia, animate images and pretty much everything else.

Scripting Language v/s Programming Language

Programming Languages are the ones which gets compiled, means that complete code will be converted into machine language code and then later these machine language code will run or will be executed. Ex : C, C++, JAVA

Scripting Languages are the ones which are interpreted, means the code will be executed line by line (whole code is not getting converted to machine language code), each line will be converted to machine level code and then executed and then flow will move to the next line.

About JS :

- Javascript was first used by Netscape browser.
- Earlier Javascript was known as Livescript.
- JS was introduced to make the client-side of web application interactive.
- After addition of Node.JS, JS is now also used on server-side of web application



Google chrome uses a V8 engine which is used to run the javascript code more efficiently in the browser, so that the application become more faster. For example, if there is a particular block of code running continuously in website then V8 engine will keep that specific code's compiled value separately so that for every execution compilation will not be required, we can directly use the compiled code.

ii. Add Javascript to Website

Open “Console” tab in Inspect Element. Here we can try our JS functions like

```
: alert("Welcome")  
: var a = 10; alert("a",a);
```

But we can not use console tab to write our JS code. We need to understand how to write a JS code like HTML code and then how to connect them.



Script tag is the html tag where we can start writing our JS code. This tag will be written inside body tag. Preferably we can keep this script tag at the last in body tag. So that whole HTML will be loaded first and after which JS will be added.

```
<script></script>
```

Different ways to add JS :

There are two ways to add JS in code.

1). In this way we write the whole JS code in HTML file itself.

Example:

```
<html>  
<body>  
<h1>  
    Document  
</h1>  
<script>  
    alert("Welcome") ;  
    var a = 4+6;  
    console.log("A", a) ;  
</script>  
</body>  
</html>
```

2). In this way we create a different JS file and add its path in src attribute in the HTML file to connect these files together.

Example:

```
(index.html)
<html>
<body>
<h1>
    Document
</h1>
<script src="./index.js">
</body>
</html>
```

```
(index.js)

alert("Welcome");
var a = 4+6;
console.log("A", a);
```

iii. Javascript Variables

Variables: Variable are nothing but just symbolic representation of values. These variables are important and very useful so that instead of using value everywhere in the code we will be using variable which will reduce the human errors.

```
var num = 1;
var name = "Prepbytes";
var isCapital = true;
```

Where :

var = It is a keyword which is used while declaring any variable.

num, name, isCapital = It is name or variable name that we have given for any value.

1, "Prepbytes", true = value

Declaration of a variable

There are 3 different ways to declare the variable

1>. var

2>. let

3>. const

var v/s let v/s const

Below mentioned are the differences which we can draw for var, let and const.

- 1>. var declarations are globally scoped or function scoped whereas let and const are block scoped.
- 2>. var variables can be updated and re-declared within its scope.
- 3>. let variables can be updated but can not be re-declared.
- 4>. const variable can neither be updated or re-declared.
- 5>. vars are hoisted.
- 6>. While var and let can be declared without being initialized, const must be initialized during declaration.

Examples to understand Scope of variables

Below are few examples which can explain how variables are declared and initialized.

Example:1

Code =

```
console.log(marks);
```

Output =

Error: Reference Error, marks is not defined

Example:2

Code =

```
var marks;  
console.log(marks);
```

Output =

undefined

Example:3

Code =

```
var marks, percentage;  
marks = 85;  
percentage = (marks * 100)/100;  
console.log(marks);  
console.log(percentage);
```

Output =

```
85  
85
```

Example:4

Code =

```
var marks, percentage, grade;
marks = 85;
percentage = (marks * 100)/100;
console.log(marks);
console.log(percentage);
if(marks>80) grade="Distinction";
console.log(grade);
grade = 12;
console.log(grade);
```

Output =

```
85
85
Distinction
12
```



In other languages like C , C++ or JAVA when we declare any variable we give a datatype like (int a = 10, where int is a datatype and a can only contain integer value now), but in JS this is not the case . Once you have declared any variable you can change its value and datatype too. Check example 4 above.

Example:5

Code =

```
let marks, percentage, grade;
marks = 85;
percentage = (marks * 100)/100;
console.log(marks);
console.log(percentage);
if(marks>80) grade="Distinction";
console.log(grade);
grade = 12;
console.log(grade);
```

Output =

```
85
85
Distinction
12
```



If you carefully notice example 4 and 5, you will understand that the variables which are declared using let and var , their values can be changed throughout the code anytime. But that is not the same case with const, check example 6 below.

Example:6

Code =

```
const marks, percentage, grade;  
marks = 85;  
percentage = (marks * 100)/100;  
console.log(marks);  
console.log(percentage);  
if(marks>80) grade="Distinction";  
console.log(grade);  
grade = 12;  
console.log(grade);
```

Output =

Error: Syntax error , missing initializer in const declaration

Example:7

Code =

```
const marks = 25;  
console.log(marks);
```

Output =

25

Example:8

Code =

```
const marks = 25;  
marks = 85;
```

Output =

Error: Type error , assignment to constant variable

Example 8 means we can not change the value of a variable which is declared using const .

Example:9

Code =

```
var marks = 100;  
var marks;
```

Output =

100

Example:10

Code =

```
let marks = 100;  
let marks;
```

Output =

Error: Syntax error, Identifier “marks” has already been declared.



Example 9 and 10 explains that if you are redeclaring any variable which means creating two variables with same name then, if you are declaring with var it will be allowed but in case you are declaring using let then it will not allow redeclaration and will throw error.

Example:11

Code =

```
const marks = 100;  
const marks;
```

Output =

Error: Syntax error, Identifier “marks” has already been declared.

Example:12

Code =

```
var marks = 100;  
let marks;
```

Output =

Error: Syntax error, Identifier “marks” has already been declared.

Scope : There are two types of variables i.e. local variables and global variables. When you are declaring a variable in a particular block of code then at which and all places this variable will be accessible, this concept is known as scope of a variable.

Example:13

Code =

```
var marks = 100;           //global-scope
if(true){
    console.log(marks);
}

function scope(){          //function-scope
    console.log(marks);
}

{                           //block-scope
    console.log(marks);
}

scope();
```

Output =

```
100
100
100
```



If we declare the variable “marks” in above example using let and const, we will get the same output.

Example:14

Code =

```
let marks = 100;
if(true){
    console.log(marks);
}
function scope(){
let marks = 200;
    console.log(marks);
}
scope();
```

Output =

```
100
200
```


iv. Hoisting

Example:1

Code =

```
console.log(marks);
var marks = 100
console.log(marks);
```

Output =

```
undefined
100
```

When you write code in JS, during execution firstly memory will be allocated to all the variables secondly code execution will happen. So for the first line we will not get any error because the variable is available in memory, but it is getting initialized in the second line so output will come correctly at the third line.

This concept is called Hoisting in JS.

So to avoid such “undefined” issues, variables should always be initialized at the top of the file.

Hoisting phenomena works only with “var” not with “let and const”.

Example:2

Code =

```
console.log(marks);
let marks = 100
console.log(marks);
```

Output =

Error : Cannot access “marks” before initialization

v. Variables Naming Convention

There are certain rules which we have to follow while naming the variables:

- Variable names can consist only of letters (a-z) (A-Z) numbers (0-9), dollar symbols (\$) and underscore (_).

Ex : ab, ab2, ab\$, ac_ (Correct)
ab*, ab/ (it will throw error)

- Variable names can't contain whitespace characters (tab or space).
- A Javascript identifier must start with a letter, underscore and dollar sign.

Ex : `cost_price`, `costPrice`, `$cost_price`, `_costPrice` (Correct)
`cost price`, `2costPrice` (it will throw error)

There are several keywords which cannot be used as the name of a variable.
 Variable names are case sensitive.

Ex : `var let` (Correct)
`var var` (it will throw error)

Keywords reserved for future

- `break`
- `case`
- `catch`
- `class`
- `const`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `export`
- `extends`
- `finally`
- `for`
- `function`
- `if`
- `import`
- `in`
- `instanceof`
- `new`
- `return`
- `super`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`
- `yield`
- `enum`

Keywords reserved of ES6

- `implements`
- `interface`
- `let`
- `package`
- `private`
- `protected`
- `public`
- `static`
- `yield`

Keywords reserved of ES6

- `await`

Keywords reserved only if it is not used in current module



“var let = 10” will work if we are not in strict mode. But it will throw error if we are in strict mode.

vi. Operators Overview & Arithmetic Operators

There are many types of operators available in JS as below :

1. Arithmetic Operator
2. Assignment Operators
3. Comparison Operators
4. Bitwise Operators
5. String Operators
6. Logical Operators
7. Ternary Operators
8. Comma Operators

1. Arithmetic Operators : Operators which are used for basic mathematical operations.

Example:1

Code =

```
var num1, num2, result;  
num1 = 2;  
num2 = 30;  
  
result = num1 + num2;  
console.log(result);  
  
result = num1 - num2;  
console.log(result);  
  
result = num1 * num2;  
console.log(result);  
  
result = num2 / num1;  
console.log(result);  
  
result = num2 % num1;  
console.log(result);
```

Output =

32
-28
60
15
0
900

vii. Comparison Operators

Below are the few comparison operators available in JS :

- Equal to : ==
- Strict Equal to : ===
- Greater than : >
- Less than : <
- Greater than equal to : >=
- Less than equal to : <=
- Not equal to : !=
- Strict not equal to : !==

- Equal to : only checks value
- Strict Equal to : checks value as well as their data type

Example:1

Code =

```
var num1, num2;
num1 = 30;
num2 = '30';
var num1, num2;

console.log(num1 == num2);
console.log(num1 === num2);

num1 = 30;
num2 = 35;

if(num1 > num2){
    console.log(num1+"is greater than"+num2);
}
else{
    console.log(num2+"is greater than"+num1);
}

num1 = 1995;
if(num1 >= 1995) console.log("Valid");
```

Output =

```
true
false
35 is greater than 30
Valid
```

Note : Statements(ex : num1>num2) which returns some value like true or false are known as expressions in Javascript.

Example:2

Code =

```
var num1, num2;
num1 = '30';
num2 = 30;

if(num1 != num2){
    console.log("Not Equal");
}
if(num1 !== num2){
    console.log("Not Strict Equal");
}
```

Output =

```
Not Strict Equal
```

Note : “ !== ” expression checks variables based on value and datatype both, so will give true result because both variables are not equal based on data types.

viii. Logical Operators

Below are few logical operators available in JS:

- Logical And : &&
- Logical Or : ||
- Logical Not : !

Logical Not (!) = It just converts the boolean value from true to false or false to true.

Logical And (&&) =

Result = A && B

A	B	Result
1	1	1
0	1	0
1	0	0
0	0	0

Logical Or (||) = When we have to check if multiple conditions are true or not

Result = A || B

A	B	Result
1	1	1
0	1	1
1	0	1
0	0	0

Example:1

Code =

```
var result = true;
var newResult = !result;
console.log(newResult)
console.log(result)
```

Output =

```
false
true
```

Example:2

Code =

```
var maths, science, cutoff, result;
maths = 85;
science = 95;
cutoff = 80;
if((maths >= cutoff) && (science >= cutoff)){
    result=true;
    console.log(result);
}

maths=60;
if((maths >= cutoff) || (science >= cutoff)){
    result=true;
    console.log(result);
}
```

Output =

true
true

ix. Logical Operators

Below are few logical operators available in JS:

condition ? Code 1 : Code 2

Example:1

Code =

```
var marks = 40, grade, cutOff = 60;
marks >= cutOff ? console.log("Great Job") : console.log("Good Job");
grade = marks >= cutOff ? 'A' : 'B';
console.log(grade);
```

Output =

Good Job
B

String Operator : When you want to concatenate to strings

Example:2

Code =

```
var myString = "Prep";
myString += "Bytes";
console.log(myString + "." + 6)
```

Output =

PrepBytes.6

ix. Logical Operators

There are two of using Increment and Decrement Operators :

- Postfix increment : a++
- Prefix increment : ++a
- Postfix Decrement : a--
- Prefix Decrement : --a

In Postfix, first the value will be used and then increment / decrement operation will happen.

In Prefix, first the increment / decrement operation will happen and then the value will be used.

Example:1

Code =

```
let a = 6;
let x = a++;
console.log(x + " " + a);
x = ++a;
console.log(x + " " + a);
```

Output =

```
6    7
8    8
```

Example:2

Code =

```
let a = 6;
let x = a--;
console.log(x + " " + a);
x = --a;
console.log(x + " " + a);
```

Output =

```
6    5
4    4
```

xi. Bitwise Operators - I

In case of Bitwise operators, first the numbers will get converted to bits and then the operation will be performed.

Below mentioned are the bitwise operators.

- Bitwise And - $a \& b$
- Bitwise OR - $a | b$ (either of the bit is 1 then o/p will be 1)
- Bitwise XOR - $a \wedge b$ (only either bit should be 1)
- Bitwise Not - $\sim a$
- Left Shift - $a \ll b$
- Sign Propagating Right Shift - $a \gg b$
- Zero fill Right Shift - $a \ggg b$

Example :

```
a = 9
b = 8
a & b
```

a will be converted to bits a (00001001) and b will also be converted to bits b (00001000). Now the operation will be performed, result will be 8(00001000).

Example:1

Code =

```
const a = 8;
const b = 9;
console.log(a&b);
console.log(a|b);
console.log(a^b);
console.log(~a);
console.log(~b);
```

Output =

```
8
9
1
-9
-10
```

xii. Bitwise Operators - ||

Example:1

Code =

```
const a = 5;
const b = 2;
console.log(a << b);
console.log(a >> b);
console.log(a >>> b);
```

Output =

```
20
1
1
```

xiii. Decision Making

Example:1

Code =

```
let weather = "raining";
if(weather === "raining"){
    console.log("No Party");
}else{
    console.log("Party");
}
```

Output =

```
20
1
```

Example:2

Code =

```
var marks = 50, grade
if(marks >= 80){
    console.log("Grade A");
}else if (marks >= 60 && marks < 80){
    console.log("Grade B");
}else{
    console.log("Fail");
}
```

Output =

Grade B



If you are having more than five conditions then instead of using if-else condition you should use switch statement.

xiv. Switch Statement



The data type of the “num” and the expression we are adding in case statement “1” should be exactly same in switch statement.

Switch statements are comparatively faster than if-else conditions but still they should be used only when you are having multiple conditions to check.

Break statement is used in switch statement so that whenever you find the happy case or scenario then rest other cases will not be performed.

Example:1

Code =

```
let num, dayOfWeek;
num = 2;
switch(num){
  case 1:
    dayOfWeek = "Monday";
    break;
  case 2:
    dayOfWeek = "Tuesday";
    break;
  case 3:
    dayOfWeek = "Wednesday";
    break;
  case 4:
    dayOfWeek = "Thursday";
    break;
  case 5:
    dayOfWeek = "Friday";
    break;
  case 6:
    dayOfWeek = "Saturday";
    break;
  case 7:
    dayOfWeek = "Sunday";
    break;
  default:
    dayOfWeek = "Invalid";
}
console.log(dayOfWeek);
```

Output =

Invalid

xv. Iteration Statements

When you such scenario where you have to do a same thing multiple times ex: Printing numbers from 1 to 10. Then in such cases we will be using loops which are known as iteration statements.

Majorly used loops are mentioned below :

- while loop
- for loop
- do while loop

For loop :

```
for(initialization ; condition ; increment/decrement){  
    code.....  
}
```

Initialization happens only for the first time when loop is running , next condition will be checked and then increment or decrement will happen and after that the code written inside the block will be executed.

Example:1

Code =

```
for(var i = 10; i<=20; i++){  
    console.log(i);  
}  
console.log("Loop terminated");
```

Output =

```
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
Loop terminated
```

Example:2

Code =

```
var i = 10;
for(; i<=20 ;){
    console.log(i);
    i++;
}
console.log("Loop terminated");
```

Output =

```
10
11
12
13
14
15
16
17
18
19
20
Loop terminated
```

We can keep the initialization and condition outside for loop also just like above example 4. But all these 3 things are must to kept means if you do not include condition in your code then you will run in an infinite loop and then finally you will have memory crash issue.

So, it is advised to keep all the three things in the for loop only.

While loop :

The structure of while loop is as follows :

```
initialization;
while (condition) {
    code....
    increment/decrement
}
```

Example:2

Code =

```
var i = 10;
while(i<=20){
    console.log(i);
    i++;
}
console.log("Loop terminated");
```

Output =

```
10
11
12
13
14
15
16
17
18
19
20
Loop terminated
```

Do-While loop :

Example:3

Code =

```
var i = 10;
do{
    console.log(i);
    i++;
}while(i<=20)
console.log("Loop terminated");
```

Output =

```
10
11
12
13
14
15
16
17
18
19
20
Loop terminated
```

Now the difference between all three loops are do-while is exit control loop and while, for loops are entry control loops. Means in case of do-while first the body gets executed and then the condition will be checked, whereas for other two loops the code will not be executed before condition is checked.

Break v/s Continue

If you are using Break statement(break;) inside a loop then it will terminate the loop. If you are using Continue statement (continue;) then that specific iteration will be skipped and rest iteration will work.

Example:4

Code =

```
for(var i = 10; i<=20; i++){  
    console.log(i);  
    if(i==15) break;  
}  
console.log("Loop terminated");
```

Output =

```
10  
11  
12  
13  
14  
15  
Loop terminated
```

Example:5

Code =

```
for(var i = 10; i<=20; i++){  
    if(i==15) continue;  
    console.log(i);  
}  
console.log("Loop terminated");
```

Output =

```
10  
11  
12  
13  
14  
16  
17  
18  
19  
20  
Loop terminated
```

Nested Loops :

Writing loop inside a loop

Example:6

Code =

```
for(var i = 1; i<=4; i++){  
    for(var j = 1; j<=3; j++){  
        console.log(i);  
    }  
}  
console.log("Loop terminated");
```

Output =

```
1  
1  
1  
2  
2  
2  
3  
3  
3  
4  
4  
4
```

We can use break; and continue statements in nested loops also. If you using break statement in inner loop then only inner loop will break but outer loop will work similar will happen with continue also.

xvi. Functions

When do we need functions ???

Functions are required when you do not want to write same block of code again and again. So we write a function which will perform specific task written inside it.

Example:1

// plain function

Code =

```
function calculateArea(){ //defining a function
    console.log("Area is 500");
}

calculateArea(); //calling a function
```

Output =

Area is 500

Example:2

// function with parameters

Code =

```
function calculateArea(length, breadth){
    let area = length * breadth;
    console.log("Area of rectangle is ", area);
    if(area > 500){
        console.log("Big rectangle");
    }else{
        console.log("Small rectangle");
    }
}

calculateArea(20, 30);
calculateArea(50, 10);
```

Output =

```
Area of rectangle is 600
Big rectangle
Area of rectangle is 500
Small rectangle
```

Example:3

// function with return statement

Code =

```
function calculateArea(length, breadth){
    let area = length * breadth;
    console.log("Area of rectangle is ", area);
    if(area > 500){
        console.log("Big rectangle");
    }else{
        console.log("Small rectangle");
    }
    return area;
}

let area = calculateArea(20, 30);
console.log("Area", area);
```

Output =

```
Area of rectangle is 600
Big rectangle
Area 600
```

Example:4

// ES6 arrow functions

Code =

```
let areaRectangle = (len, br) =>
  console.log("Length is " + len + "& breadth is " + br);

areaRectangle(10, 20);
```

Output =

```
Length is 10 & breadth is 20
```

Data structure overview

◇ What is Data-Structure and How it is important?

Data Structure can be defined as the collection of data objects which provides a way of storing and managing data in the computer so that it can be used. Various Data Structures types are arrays, Stack, Queue, etc.

Example:

Like Many of you been scolded by your parents for not being able to find a particular outfit in your messy wardrobe?

Yes, right? Your parents give the advice to keep your clothes in a systematic manner so the next time you want something you can easily pick it up.

In this situation, you need to arrange & keep your clothes (data) in such a structure that when searching for a particular thing you don't have to hassle much.

◇ What is collection and explain its types?

We have different data structure implemented and we have different methods implemented we just have to know how to create objects of those data structures.

How to initialize those data structure and how to go ahead and call the method of those data structure.

what are different methods that are being supported. And we can pass the parameter and what sort of parameters we can pass.

There are two types of collection: -

- **Indexed-collection:-** In JavaScript, an array is an indexed collection. An array is an ordered set of values that has a numeric index.

Example: An array called 'student' that contains the name of the students and the index values are the Roll Numbers of the students. JavaScript does not have an explicit array data type. However, we can use the predefined Array object in JavaScript and its methods to work with arrays.

- **Keyed-Indexing:-** Keyed collections are data collections that are ordered by key not index. They are associative in nature. Map and set objects are keyed collections and are iterable in the order of insertion.

◇ Explain array and different ways we can define array?

An array is a special variable, which can hold more than one value at a time.

An array can hold many values under a single name, and you can access the values by referring to an index number.

There are different ways we can define a array:

Let arr=new Array(element0, element1,.....element)

Let arr=Array(element0, element1.....element)

Let arr= [element1,element2,element3]

Let arr=new Array(arrayLength)

Let arr=Array(arrayLength)

Let arr=[]

◇ Explain How we iterate in an array and its different ways?

Array iteration methods operate on every array item. Let's see various iteration methods on arrays.

- **Array.forEach method:** The `forEach()` method calls a function (a callback function) once for each array element.

Input:

```
const array1 = ['a', 'b', 'c'];  
array1.forEach(element => console.log(element));
```

Output:

a b c

Note that the function takes 3 arguments:

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

- **For Loop:** In this we can traverse in our array through for loop.

```
<script>  
array = [ 1, 2, 3, 4, 5, 6 ];  
for (index = 0; index < array.length; index++) {  
    console.log(array[index]);  
}  
</script>
```

- **Using every method:** The `every()` method checks if all elements in an array pass a test (provided as a function).

```
<script>  
const isBelowThreshold = (currentValue) => currentValue < 40;  
  
const array1 = [1, 30, 39, 29, 10, 13];  
  
console.log(array1.every(isBelowThreshold));</script>
```

- **Using map :-** A map applies a function over every element and then returns the new array.

```
<script>
index = 0;
array = [ 1, 2, 3, 4, 5, 6 ];
square = x => Math.pow(x, 2);
squares = array.map(square);
console.log(array);
console.log(squares);</script>
```

output:

```
1 2 3 4 5 6
1 4 9 16 25 36
```

◇ Explain various Array's method?

1. join() method: The join() method also joins all array elements into a string. It behaves just like toString(), but in addition you can specify the separator:

Input:

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

```
<h2>JavaScript Array Methods</h2>
```

```
<h2>join()</h2>
```

```
<p>The join() method joins array elements into a string.</p>
```

```
<p>In this example we have used " * " as a separator between the elements:</p>
```

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

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>
</body>
</html>
```

Output:

JavaScript Array Methods

join()

The join() method joins array elements into a string.

It this example we have used " * " as a separator between the elements:

Banana * Orange * Apple * Mango

2. pushing method: The push() method adds a new element to an array (at the end):

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Array Methods</h2>
```

```
<h2>push()</h2>
```

```
<p>The push() method returns the new array length.</p>
```

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

```
<p id="demo1"></p>
```

```
<p id="demo2"></p>
```

```
<script>
```

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

```
document.getElementById("demo1").innerHTML = fruits;
```

```
function myFunction() {
```

```
    document.getElementById("demo1").innerHTML = fruits.push("Kiwi");
```

```
    document.getElementById("demo2").innerHTML = fruits;
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

JavaScript Array Methods

push()

The push() method returns the new array length.

Try it

12

Banana,Orange,Apple,Mango,Kiwi,Kiwi,Kiwi,Kiwi,Kiwi,Kiwi,Kiwi,Kiwi

3. Popping method: The pop() method removes the last element from an array:

Input:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Array Methods</h2>
```

```
<h2>pop()</h2>
```

```
<p>The pop() method removes the last element from an array.</p>
```

```
<p id="demo1"></p>
```

```
<p id="demo2"></p>
```

```
<script>
```

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

```
document.getElementById("demo1").innerHTML = fruits;
```

```
fruits.pop();
```

```
document.getElementById("demo2").innerHTML = fruits;
```

```
</script>
```

```
</body>
```

```
</html>
```

Output =

JavaScript Array Methods

pop()

The pop() method removes the last element from an array.

Banana,Orange,Apple,Mango

Banana,Orange,Apple

4. Shifting method: Shifting is equivalent to popping, working on the first element instead of the last. The shift() method removes the first array element and "shifts" all other elements to a lower index.

Input:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Array Methods</h2>
```

```
<h2>shift()</h2>
```

```
<p>The shift() method removes the first element of an array (and "shifts" all other elements to the left):</p>
```

```
<p id="demo1"></p>
```

```
<p id="demo2"></p>
```

```
<script>
```

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

```
document.getElementById("demo1").innerHTML = fruits;
```

```
fruits.shift();
```

```
document.getElementById("demo2").innerHTML = fruits;
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

JavaScript Array Methods

shift()

The shift() method removes the first element of an array (and "shifts" all other elements to the left):

Banana,Orange,Apple,Mango

Orange,Apple,Mango

5. Unshifting method: The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Input:

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

<h2>JavaScript Array Methods</h2>
<h2>unshift()</h2>
<p>The unshift() method adds new elements to the beginning of an array.</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.unshift("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Output:

```
JavaScript Array Methods
unshift()
The unshift() method adds new elements to the beginning of an array.

Try it
Lemon,Lemon,Lemon,Banana,Orange,Apple,Mango
```

In this above example you can see I have clicked 3 times on try it.

6. Concatenating method:- The concat() method creates a new array by merging (concatenating) existing arrays:

Input:

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

<h2>JavaScript Array Methods</h2>
<h2>concat()</h2>
<p>The concat() method is used to merge (concatenate) arrays:</p>

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

<script>
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

Output:

```
JJavaScript Array Methods
concat()
The concat() method is used to merge (concatenate) arrays:

Cecilie,Lone,Emil,Tobias,Linus
```

7. Splicing method:- The splice() method can be used to add new items to an array:

Input:

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

<h2>JavaScript Array Methods</h2>
<h2>splice()</h2>
<p>The splice() method adds new elements to an array.</p>

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

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br>" + fruits;
"Original Array:<br>" + fruits;

function myFunction() {
  fruits.splice(2, 0, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
}
</script>

</body>
</html>
```

Output:

JavaScript Array Methods

splice()

The splice() method adds new elements to an array.

Try it

Original Array:

Banana,Orange,Apple,Mango

New Array:

Banana,Orange,Lemon,Kiwi,Apple,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.

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

Input:

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

<h2>JavaScript Array Methods</h2>
<h2>splice()</h2>
<p>The splice() methods can be used to remove array elements.</p>

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

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.splice(0, 1);
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Output:

```
JavaScript Array Methods
splice()
The splice() methods can be used to remove array elements.
```

```
Try it
Orange,Apple,Mango
```

I have clicked try it once and it removed Banana.

8. Slicing method: The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

The slice() method creates a new array. It does not remove any elements from the source array. The splice() method can be used to add new items to an array:

Input:

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

<h2>JavaScript Array Methods</h2>
<h2>slice()</h2>
<p>This example slices out a part of an array starting from array element 1 ("Orange"):</p>

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

<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

Output:

```
JavaScript Array Methods
slice()
```

This example slices out a part of an array starting from array element 1 ("Orange"):

```
Banana,Orange,Lemon,Apple,Mango
```

```
Orange,Lemon,Apple,Mango
```

9. Sort method: The sort() method sorts an array.

(i) Alphabetic sort(): The sort() method sorts an array alphabetically.

Input:

```
<script>
// JavaScript to illustrate sort() function
function func() {

    // Original string
    var arr = ["Geeks", "for", "Geeks"]

    document.write(arr);
    document.write("<br>");
    // Sorting the array
    document.write(arr.sort());
}
func();
</script>
```

Output:

```
Geeks,for,Geeks
Geeks,Geeks,for
```

(ii) Numeric sort: By default, the sort() function sorts values as strings.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce incorrect result when sorting numbers.

Input:

```
<script>
// JavaScript to illustrate sort() function
function func() {

    //Original string
    var arr = [2, 5, 8, 1, 4]

    //Sorting the array
    document.write(arr.sort());
    document.write("<br>");
    document.write(arr);
}
func();
</script>
```

Output:

1,2,4,5,8

1,2,4,5,8

10. Reverse method: The reverse() method reverses the elements in an array.

Input:

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

<h2>JavaScript Arrays</h2>

<p>The Array.reverse() method reverses the order of the elements in an array.</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.reverse();
</script>

</body>
</html>
```

Output:

JavaScript Arrays

The Array.reverse() method reverses the order of the elements in an array.

Mango,Apple,Orange,Banana

11. Indexof method:

The indexOf() method searches an array for a specified item and returns its position.

The search will start at the specified position (at 0 if no start position is specified), and end the search at the end of the array.

indexOf() returns -1 if the item is not found.

If the item is present more than once, the indexOf method returns the position of the first occurrence.

Input:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Arrays</h2>
```

```
<p>The Array.indexOf() method searches an array for a specified item and returns its position.</p>
```

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

```
<script>
```

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

```
document.getElementById("demo").innerHTML = fruits.indexOf("Apple");
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

JavaScript Arrays

The Array.indexOf() method searches an array for a specified item and returns its position.

2

12. LastIndexOf() method: The lastIndexOf() method returns the last index (position) of a specified value.

The search starts at a specified position (at the end if no start position is specified), and ends the search at the beginning of the array.

lastIndexOf() returns -1 if the item is not found.

If the search value is present more than once, the method returns the position of the last occurrence.

Input:

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

<h2>JavaScript Arrays</h2>

<p>Array.lastIndexOf() returns the last position of a specified value.</p>

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

<script>
const fruits = ["Apple", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.lastIndexOf("Apple");
</script>

</body>
</html>
```

Output:

JavaScript Arrays
Array.lastIndexOf() returns the last position of a specified value.

13. filter Method: The filter() method creates an array filled with all array elements that pass a test (provided by a function).

filter() does not execute the function for empty array elements.

filter() does not change the original array.

Input:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Arrays</h2>
```

```
<p>Click the button to get every element in the array that has a value of 18 or more.</p>
```

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

```
<script>
```

```
const ages = [32, 33, 16, 40];
```

```
document.getElementById("demo").innerHTML = ages.filter(checkAdult);
```

```
function checkAdult(age) {
```

```
  return age >= 18;
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

JavaScript Arrays

Click the button to get every element in the array that has a value of 18 or more.

32,33,40

◇ What is map and how to use maps in javascript?

- Map is a data structure in JavaScript which allows storing of [key, value] pairs where any value can be either used as a key or value.
- The keys and values in the map collection may be of any type and if a value is added to the map collection using a key which already exists in the collection, then the new value replaces the old value.
- The iteration of elements in a map object is done in the insertion order and a “for...” loop returns an array of all [key, value] pairs for each iteration.

◇ how we create object of map?

```
let map1 = new Map();
```

◇ how we set value corresponding to a key?

```
map1.set('a', 1);  
map1.set('b', 2);  
map1.set('c', 3);
```

there is one another method with which we can add value to corresponding key:

```
let cityMap=new Map([  
  ['gurgaon','haryana'],  
  ['chennai','Tamil Nadi'],  
  ['jamshedpur','jharkhand']  
])
```

◇ How to get value corresponding to a key in a map?

```
console.log(map1.get(key));
```

◇ How to find that some key is present or not in our map?

Input:

```
const map1 = new Map();  
map1.set('bar', 'foo');  
  
console.log(map1.has('bar'));  
console.log(map1.has('baz'));
```

Output:

```
› true  
› false
```

◇ Explain ways to traverse in a map?

• Using for loop:

```
for (let [key, value] of map)  
{  
  console.log(key + " = " + value);  
}
```

• Using foreach():

Input:

```
<script>  
  
  let mp=new Map()  
  
  mp.set("a",1);  
  mp.set("b",2);  
  mp.set("c",3);  
  
  mp.forEach((values,keys)=>{  
    document.write(values,keys+"<br>")  
  })  
</script>
```

- **map.entries()** - Returns an iterable of key,value

Input:

```
let map = new Map()
```

```
map.set("one", "first element");  
map.set("two", "second element");  
map.set(3, "third element");
```

Output:

```
one = first element  
two = second element  
3 = third element
```

◇ What is set in java script?

A set is a collection of items which are unique i.e no element can be repeated.

```
// it contains  
// ["sumit","amit","anil","anish"]  
var set1 = new Set(["sumit","sumit","amit","anil","anish"]);  
  
// it contains 'f', 'o', 'd'  
var set2 = new Set("fooooooooood");  
  
// it contains [10, 20, 30, 40]  
var set3 = new Set([10, 20, 30, 30, 40, 40]);  
  
// it is an empty set  
var set4 = new Set();
```

◇ How to check that we have a value in our set or not?

```
// creating set
var set1 = new Set();

// adding element to the set
set1.add(50);
set1.add(30);

// prints true
console.log(set1.has(50));

// prints false
console.log(set1.has(10));
```

◇ How we can delete a value from the set?

Input:

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

<script>
var set = new Set();
set.add("jQuery");
set.add("AngularJS");
set.add("Bootstrap");
document.writeln("Size before invoking delete() method: "+ set.size+"<br>");
set.delete("Bootstrap");
document.writeln("Size after invoking delete() method: "+set.size);
</script>

</body>
</html>
```

Output:

Size before invoking delete() method: 3

Size after invoking delete() method: 2

◇ Explain ways to traverse in an set?

- **Using for loop:**

Input:

```
let set = new Set();
set.add(1);
set.add(2);
set.add(3);
set.add(4);
for (let item of set) {
  console.log(item);
}
```

Output:

```
1
2
3
4
```

- **Using enteries:-**

Input:

```
let set = new Set();
set.add(1);
set.add('prashant');
set.add({a: 1, b: 2, c: 3, d: 4});

for (let [key, value] of set.entries()) {
  console.log(key, value);
}
```

Output:

11

prashant prashant

{a: 1, b: 2, c: 3, d: 4} {a: 1, b: 2, c: 3, d: 4}

• Using for each():-**Input:**

```
let set = new Set();
set.add(1);
set.add(2);
set.add(3);
set.add(4);
//foreach loop
set.forEach((e) => {console.log(e);});
```

Output:

1
2
3
4

◇ What is DOM?

DOM stands for Document Object Model.

›When a web page is loaded, the browser creates a Document Object Model of the page.

›The HTML DOM is a standard object model and programming interface for HTML.

›In the DOM, all HTML elements are defined as objects.

›DOM represents the documents as objects.

›everything you can see such as buttons, headings etc they will be represented in objects

◇ What are Objects?

- › Every thing surrounds you is an object such as a person, a car etc.
 - › Every object has some properties and some methods.
 - › For example a cup has its design, color, material.
 - › A person has name, occupation as properties.
 - › Methods such as bio, teach.
 - › Suppose we have heading in HTML we can its font size, font color, etc. as its properties.
- this is how we make objects:

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

This is how you can access the one or more objects or elements on our webpage:

```
Document.firstChild
```

◇ How to access Elements DOM elements?

Input:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>document</title>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <meta name="description" content="" />
</head>
<body>
```

```
<h1 class="main-heading">Hello, world!</h1>
<button>Click Me</button>
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
</ul>
</body>
</html>
```

- `document.getElementsByTagName("Li")` -->this will give array of list.
- `document.getElementsByTagName("h1")` -->this will give you the heading.
- `document.getElementsByTagName("h1")[0].style.color`-->this is how you can access the first element and color. //0:h1
- In case get element by tag name it will give list of all the elements that have particular tag as written above. And individual element we can access with the help of indexing.
- we can also get elements by class.
 - >`document.getElementsByTagName("class name here")`--> it will give all the elements which have this class.
- we can also get elements by id.
 - >`document.getElementsByTagName("id name here")` ---->it will give element which have this id .*this will not give list because id is unique.
- we can also use `querySelector`.
 - >for example there are two buttons in your document .it will specifically choose first one
 - >syntax:`document.querySelector("button");`
 - >`document.querySelectorAll("button")[1];` -->this will give second button
 - >`document.querySelector(".list-item");`-->this will give only first element of list

›document.querySelector(".list-item");-->this will give only first element of list
 ›document.querySelectorAll(".list-item");-->this will give all element of list.
 ›document.querySelectorAll("#id name here");-->give elements of this id.
 ›suppose there are two list with class id list and secondary list and we have to access or change in list with class id list.
 syntax:document.querySelector(".list.list-item");
 ›suppose we have same class for paragraph and ordered list and we want to change or access the paragraph.
 syntax:document.querySelector("p.text-item");

◇ How we can access and manipulate the elements?

Input:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Accessing Elements in the DOM</title>
```

```
<style>
```

```
html { font-family: sans-serif; color: #333; }
```

```
body { max-width: 500px; margin: 0 auto; padding: 0 15px; }
```

```
div, article { padding: 10px; margin: 5px; border: 1px solid #dedede; }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>Accessing Elements in the DOM</h1>
```

```
<h2>ID (#demo)</h2>
```

```
<div id="demo">Access me by ID</div>
```

```
<h2>Class (.demo)</h2>
```

```
<div class="demo">Access me by class (1)</div>
```

```
<div class="demo">Access me by class (2)</div>
```

```
<h2>Tag (article)</h2>
```

```
<article>Access me by tag (1)</article>
```

```
<article>Access me by tag (2)</article>
```

```
<h2>Query Selector</h2>
```

```
<div id="demo-query">Access me by query</div>
```

```
<h2>Query Selector All</h2>
```

```
<div class="demo-query-all">Access me by query all (1)</div>
```

```
<div class="demo-query-all">Access me by query all (2)</div>
```

```
</body>
```

```
</html>
```

Output:



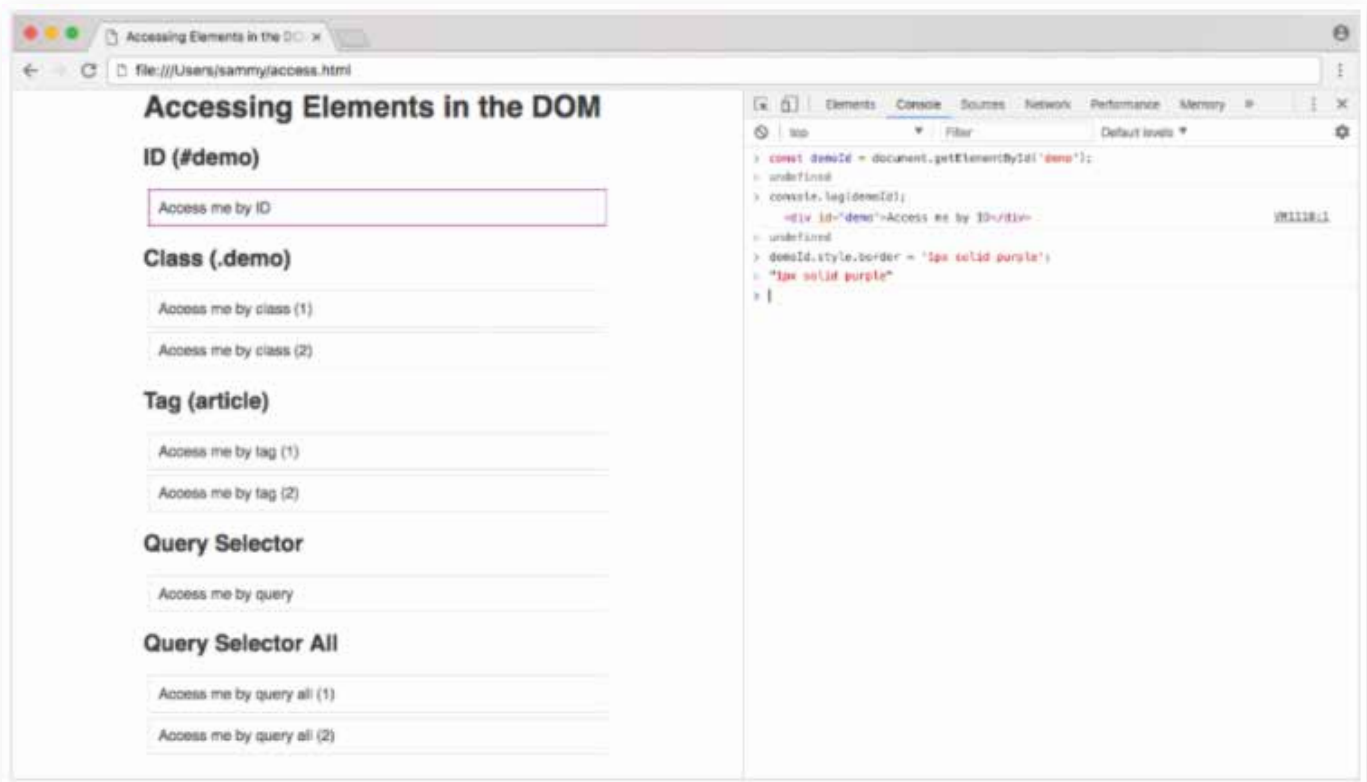
- Access element by id:

Input:

```
const demoId = document.getElementById('demo');
```

```
demoId.style.border = '1px solid purple';
```

Output:



- Access element by class:

Input:

```
const demoClass = document.getElementsByClassName('demo');
```

```
for (i = 0; i < demoClass.length; i++) {  
  demoClass[i].style.border = '1px solid orange';  
}
```

Output:



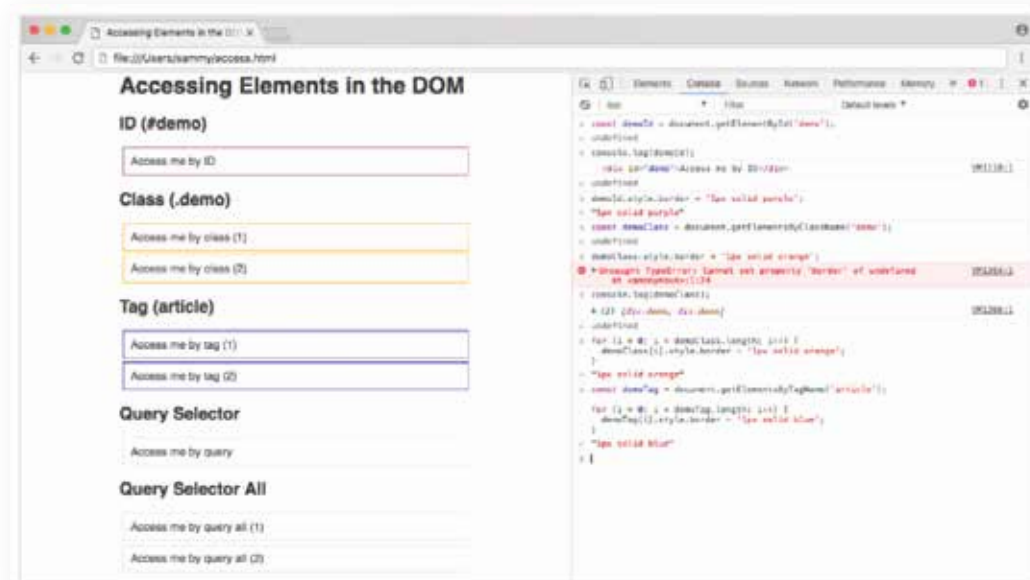
- Access element by tagname:

Input:

```
const demoTag = document.getElementsByTagName('article');
```

```
for (i = 0; i < demoTag.length; i++) {
  demoTag[i].style.border = '1px solid blue';
}
```

Output:

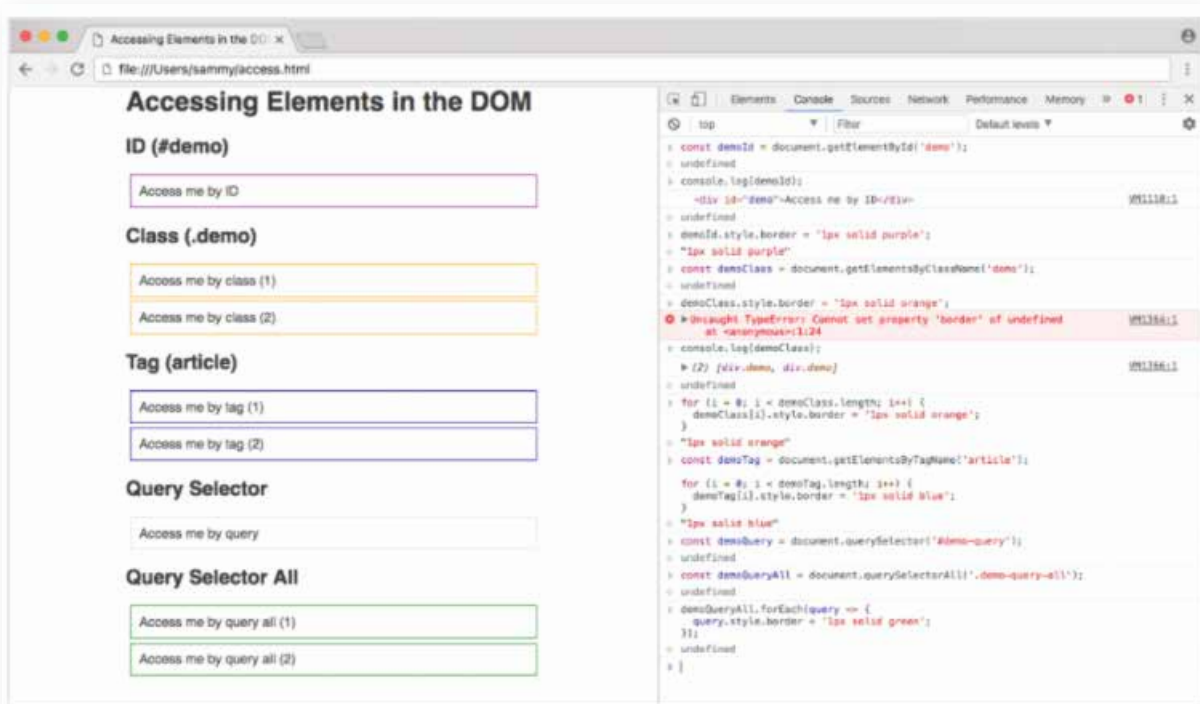


- Access element by query selectors:

Input:

```
const demoQuery = document.querySelector('#demo-query');
demoQueryAll.forEach(query => {
  query.style.border = '1px solid green';
});
```

Output:



◇ What are event listeners?

Event listeners wait for certain event to happen and as soon as event happens, they perform several tasks.

An event is an important part of JavaScript. A web page responds according to an event occurred. Some events are user generated and some are generated by API's. An event listener is a procedure in JavaScript that waits for an event to occur. The simple example of an event is a user clicking the mouse or pressing a key on the keyboard.

suppose if you want to change the background color of heading as you hover your mouse over it. this can be done by event listeners.

◇ Explain various event listeners?

- **Click event listeners**: Let's see the code for how click event listeners works.

Input:

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

<p>This example uses the addEventListener() method to execute a function when a user
clicks on a button.</p>

<button id="myBtn">Try it</button>

<p id="demo">

<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```

Output: When I have not clicked on try it.

This example uses the addEventListener() method to execute a function when a user clicks on a button.

Try it

Output: When I have clicked on try it.

This example uses the addEventListener() method to execute a function when a user clicks on a button.

Try it

Hello World

- **Mouseover and Mouseout event listener:**

Input:

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

<p>This example uses the addEventListener() method to add many events on the same button.
</p>

<button id="myBtn">Try it</button>

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

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
    document.getElementById("demo").innerHTML += "Moused over!<br>"
}

function mySecondFunction() {
    document.getElementById("demo").innerHTML += "Clicked!<br>"
}

function myThirdFunction() {
    document.getElementById("demo").innerHTML += "Moused out!<br>"
}
</script>

</body>
</html>
```

Output:

This example uses the addEventListener() method to add many events on the same button.

Try it

Moused over!

Clicked!

Moused out!

This is the output when I first carried my cursor over try it button it printed mouse over and when I clicked on try it it printed clicked and when I carry my mouse out from the try it. Then it printed Mouse out.

◇ What is objects in Java script?

-An object is a standalone entity, with some properties and type.

For example a cup which is a object with some properties such as {color, it's design ,weight, material}.same way JavaScript objects can have some properties which defines its characteristics.

-let's take car as an object so it's design, color, body defines its characteristics.

-A JavaScript object has properties associated with it.

-In array we access elements by index, but in objects we access value by properties.

◇ How to create object in javascript?

we can also have array in property value such as["singing","dancing"].

Code:

```
var shyam = {           //As shyam is an object and name,age all are property of it
  name: 'Shyam',        //and shyam ,23, all are properties value.
  profession: 'teacher',
  age: 23,
  hobbies:["singing","dancing"]
};
```

◇ How to access a object Property values?

There are two ways to access a object property value:

Code:

```
var shyam = {           //As shyam is an object and name,age all are property of it
  name: 'Shyam',        //and shyam ,23, all are properties value.
  profession: 'teacher',
  age: 23,
  hobbies:["singing","dancing"]
};
```

A. Dot notation:

1. `console.log(shyam.name)`
2. `console.log(shyam.profession)`

B. Bracket notation:

1. `console.log(shyam["name"])`
2. `console.log(shyam["profession"])`

◇ How to create method in object?

Code:

```
var shyam = {  
  name: 'Shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies: ["singing", "dancing"]  
  bio: function(){  
    console.log("I am fine");  
  }  
};
```

Input:

```
console.log(Shyam["bio"]  
console.log(Shyam.bio());
```

Output:

```
I am fine  
I am fine
```

◇ What is THIS in JavaScript and explain it with example?

In an object method, this refers to the "owner" of the method.

In the example on the top of this page, this refers to the person object.

The person object is the owner of the full Name method.

EXAMPLE:

Code:

```
var person = {  
  firstName: "John",  
  lastName: "Doe",  
  id: 5566,  
  bio: function() {  
    console.log("I am"+this.firstName+this.lastName)  
    //this will indicate the object (person) this.firstName-->John,this.lastName-->Doe  
  }  
};
```

Input:

```
console.log(person.bio);  
person.bio()
```

Output:

I am John Doe

◇ What are first class Functions?

When function in a language is treated like any other variable.

The language is said to be first class functions.

A function:

- a). can be passed as an argument.
- b). can be returned by another function.
- c). can be assigned as a value to a variable.

Note: In special cases when a class's private data needs to be accessed directly without using objects of that class, we need friend functions.

Note: The friend function can be a member of another class or a function that is outside the scope of the class.

EXAMPLE: EXAMPLE OF PASSING FUNCTIONS IN JAVASCRIPT AS AN ARGUMENT:

Input:

```
var temp=function(){  
    return "Sunday"  
}  
function flag(string,day){  
    console.log(string+temp())  
}  
flag("Today is",temp)
```

Output:

Today is Sunday

◇ What is higher order function explain with example?

A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

EXAMPLE: 1**Code:**

```
function even(num){  
  if(num%2==0){  
    console.log("the no"+num+"which you have entered is even")  
  }  
  else{  
    console.log("the no"+num+"which you have entered is not even")  
  }  
}
```

Input:

```
var num(5)
```

Output:

the no 5 which you have entered is not even

EXAMPLE: 2**Code:**

```
function promotionCalculator(designation){  
  if(designation=="Manager"){  
    return function(experience){  
      console.log("Promotion for"+designation+"with"+"experience"+experience+"is senior manager")  
    }  
  }  
  else{  
    return function(experience){  
      console.log("Promotion for"+designation+"with"+"experience"+experience+"is based on  
experience")  
    }  
  }  
}
```

Input:

```
val promotion=promotionCalculator("Manager");  
promotion(5): // First way  
promotionCalculator("Manager")(5); //second way
```

Output:

```
Promotion for Manager with experience 5 is senior manager  
Promotion for Manager with experience 5 is senior manager
```

◇ How we use call at the time of passing arguments in function?

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

EXAMPLE:**Code:**

```
var shyam = {  
  name: 'Shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies:["singing","dancing"]  
  workingBio:function(){  
    console.log("I am"+this.firstName+" "+this.lastName)  
  }:  
  
  var shiva= {  
    name: 'shiva',  
    profession: 'dancer',  
    age: 20,  
    hobbies:["singing","dancing"]  
  }:  
}
```

Input:

shyam.workingBio.call(shiva) // call use »this function calling workingBio with object as shiva

Output:

shiva dancer

EXAMPLE:2 ** WHEN METHODS ALSO CONTAINS SOME ARGUMENTS.

Code:

```
var shyam = {  
  name: 'Shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies: ["singing", "dancing"]  
  workingBio: function(experience, age) {  
    console.log("I am " + this.firstName + " " + this.lastName)  
  }  
};  
  
var shiva = {  
  name: 'shiva',  
  profession: 'dancer',  
  age: 20,  
  hobbies: ["singing", "dancing"]  
};
```

Input:

shyam.workingBio.call(shiva, 3, 23) // method borrowing

Output:

shiva dancer 3,23

◇ How we use apply at the time of passing arguments in function?

It also works like call the difference is that it pass the value with the help of array.

EXAMPLE:

Code:

```
var shyam = {  
  name: 'Shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies: ["singing", "dancing"]  
  workingBio: function(experience, age) {  
    console.log("I am " + this.firstName + " " + this.lastName);  
  }  
};
```

```
var shiva = {  
  name: 'shiva',  
  profession: 'dancer',  
  age: 20,  
  hobbies: ["singing", "dancing"]  
};
```

Input:

```
shyam.workingBio.apply(shiva, [3, 23])
```

Output:

```
shiva dancer 3,23
```

◇ How we use bind at the time of passing arguments in function?

As in call and apply function calling or method calling happens at the same place where we are having statement. But in bind it binds the object and we can call function anytime.

EXAMPLE:

Code:

```
var shyam = {  
  name: 'shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies: ["singing", "dancing"]  
  workingBio: function(experience, age) {  
    console.log("I am " + this.firstName + " " + this.lastName);  
  }  
};  
var john = {  
  name: "john",  
  profession: "dancer"  
}
```

Input:

```
var bio = shyam.workingBio.bind(john, 6, 25)  
bio();  
var bio = shyam.workingBio.bind(john)  
bio(6, 25)
```

Output:

```
john,dancer ,6,25  
john,dancer,6,25
```

◇ How we can use call,apply,bind when method is outside the object?

As in call and apply function calling or method calling happens at the same place where we are having statement. But in bind it binds the object and we can call function anytime.

EXAMPLE:

Code:

```
var shyam = {  
  name: 'Shyam',  
  profession: 'teacher',  
  age: 23,  
  hobbies: ["singing", "dancing"]  
};  
  
function workingbio(experience,age){  
  console.log("I am"+this.firstName+" "+this.lastName)  
}  
  
var shiva= {  
  name: 'shiva',  
  profession: 'dancer',  
  age: 20,  
  hobbies: ["singing", "dancing"]  
};
```

Input:

```
workingBio.call(shiva,3,23)
```

Output:

```
shiva dancer 3,23
```

◇ what are constructor functions?

Objects are also created with the help of constructors functions.

A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

EXAMPLE:

Code:

```
var shyam = {  
  name: 'shyam',  
  profession: 'teacher',  
  hobbies: ["jogging", "reading"]  
};
```

```
var shiva= {  
  name: 'shiva',  
  profession: 'dancer',  
  hobbies: ["singing", "dancing"]  
}
```

In these two objects we have to write properties again and again such as name profession, etc. We use constructor to avoid writing property again and again.

◇ How can we make constructors in javascript?

Objects are also created with the help of constructors functions.

A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

EXAMPLE:

Code:

```
var shyam = {
  name: 'shyam',
  profession: 'teacher',
  hobbies:["jogging","reading"]
  workingBio:function(experience,age){
    console.log(this.name+" "+this.profession+" "+experience+
      " "+age)
  };
};

var shiva= {
  name: 'shiva',
  profession: 'dancer',
  hobbies:["singing","dancing"]
  workingBio:function(experience,age){
    console.log(this.name+" "+this.profession+" "+experience+
      " "+age)
  };
};

//Constructor function
function Person(name,profession,hobbies){
  this.name=name;
  this.profession=profession;
  this.hobbies=hobbies;
```

Input:

```
var shyam=newPerson("shyam","teacher",["jogging","reading"])
var shiva=newPerson("shiva","dancer",["singing","dancing"])
console.log(shyam);
console.log(shiva);
```

Output:

```
name:"shyam"
profession:"teacher"
hobbies: ["jogging","reading"]
workingBio:f(experience,age)
>__proto__:Object
```

```
name:"shiva"
profession:"dancer"
hobbies: ["singing","dancing"]
workingBio:f(experience,age)

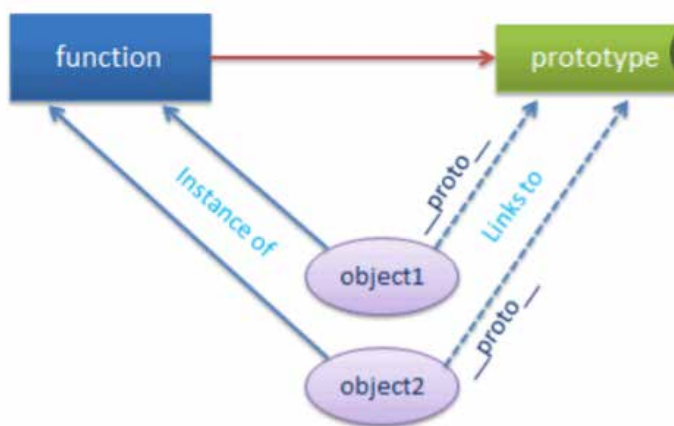
>__proto__:Object
```

So instead of writing properties again and again we simply make constructors and call it again and again for different parameters.

◇ What are prototypes?

Basically, Prototypes are mechanism by which objects can inherit properties from each other. JavaScript is a prototype based language, so, whenever we create a function using JavaScript, JavaScript engine adds a prototype property inside a function, Prototype property is basically an object (also known as Prototype object), where we can attach methods and properties in a prototype object, which enables all the other objects to inherit these methods and properties.

Every object you created have `__proto__` property will point to prototype property of constructor function.



1. Now, we will add a method `calculateAge()` to the Prototype property in a `Person` function constructor which will inherit by the different objects. Below is the code for this:-

Code:

```
<script>
// function constructor
function Person(name, job, yearOfBirth){
    this.name= name;
    this.job= job;
    this.yearOfBirth= yearOfBirth;
}
Person.prototype.calculateAge= function(){ // how we use constructor
    console.log('The current age is: '+(2019- this.yearOfBirth));
}
console.log(Person.prototype);
</script>
```

1. Now we will create 2 different objects which will inherit calculateAge() method and remember, When a certain method(or property) is called, it first checks inside the object but when it doesn't find, then search moves on Object's prototype.

Code:

```
<script>
```

```
// function constructor
function Person(name, job, yearOfBirth){
    this.name= name;
    this.job= job;
    this.yearOfBirth= yearOfBirth;
}

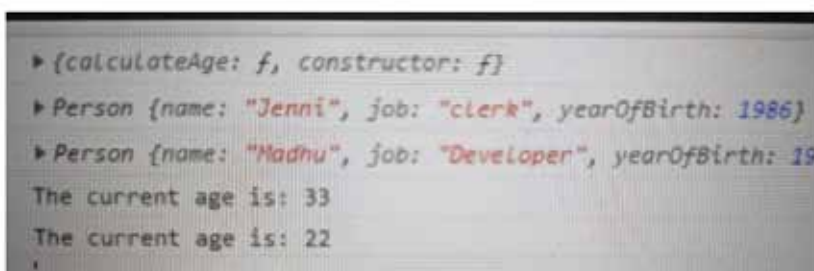
// adding calculateAge() method to the Prototype property
Person.prototype.calculateAge= function(){
    console.log('The current age is: '+(2019- this.yearOfBirth));
}

console.log(Person.prototype);

// creating Object Person1
let Person1= new Person('Jenni', 'clerk', 1986); //creating objects with the help of constructors.
console.log(Person1)
let Person2= new Person('Madhu', 'Developer', 1997);
console.log(Person2)

Person1.calculateAge();
Person2.calculateAge();

</script>
```



```

> {calculateAge: f, constructor: f}
> Person {name: "Jenni", job: "clerk", yearOfBirth: 1986}
> Person {name: "Madhu", job: "Developer", yearOfBirth: 1997}
The current age is: 33
The current age is: 22

```


Note: Here, we created two Objects Person1 and Person2 using constructor function Person, when we called

Person1.calculateAge() and Person2.calculateAge(), First it will check whether it is present inside Person1 and Person2 object, if it is not present, it will move Person's Prototype object and prints the current age, which shows Prototype property enables other objects to inherit all the properties and methods of function constructor.

◇ How can we create object using create and how use bracket{} and null in creating objects?

The Object.create() method creates a new object, using an existing object as the prototype of the newly created object.

1. Use of {}:

Syntax:

```
var object name=Object.create({});
```

Code:

```
var Shyam=Object.create(null);
```

Input:

```
console.log(shyam);
```

Output:

```
{  
  __proto__:Object
```

2. Use of null:

Syntax:

```
var Shyam=Object.create(null);
```

Code:

```
console.log(shyam);
```

Input:

```
console.log(shyam);
```

Output:

```
{}
```

There is no properties and `__proto__` property when we pass null. if we use null we don't set any properties and we also don't inherit anything.

◇ How can we add some properties and methods to object using create?

1. We can add properties like this.

Code:

```
var object name=Object.create({})
```

```
Shyam.name="Shyam";
```

```
Shyam.profession="teacher"
```

Input:

```
console.log(shyam);
```

Output:

```
name="Shyam"
```

```
profession="teacher"
```

```
__proto__:Object
```

1. We can add methods like this.**Code:**

```
var object name=Object.create({})  
Shyam.name="Shyam";  
Shyam.profession="teacher"  
shyam.workExperience = function(){  
  console.log("shyam is fine");  
};
```

Input:

```
console.log(shyam);
```

Output:

```
name="Shyam";  
profession="teacher"  
workExperience:f()  
__proto__:Object
```

◇ How can we inherit the properties of one object in other objects in javascript?

Code:

```
var shiva={  
    name: "shiva",  
    profession: "dancer"  
}  
var shyam=Object.create(shiva);
```

Input:

```
console.log(shyam);
```

Output:

```
{  
  __proto__:  
    name="shiva"  
    profession="dancer"
```

Input:

```
console.log(shyam.name)
```

Output:

```
Shiva
```

- we can also use this with methods.

Code:

```
var shiva={  
  name: "shiva",  
  profession: "dancer"  
  workExperience:function(){  
    console.log(this.name+"is experienced")  
  }  
};
```

```
var shyam=Object.create(shiva)
```

```
shyam.workExperience=function(){  
  console.log("shyam is fine")  
}
```

Input:

```
shyam.workExperience();
```

Output:

```
shyam is working hard
```