JavaScript-Basics, Function, Loop, Array

By Geeta Singh

JavaScript

JavaScript (JS) is the most popular lightweight, interpreted compiled programming language. It can be used for both **Client-side** as well as **Server-side** developments. JavaScript also known as a scripting language for web pages. Major companies like Microsoft, Uber, Google, Netflix, and Meta use JavaScript in their projects.

How JavaScript makes HTML build-website better

- JavaScript is an advanced programming language that makes web pages more interactive and dynamic whereas HTML is a standard markup language that provides the primary structure of a website.
- JavaScript simply adds dynamic content to websites to make them look good and HTML work on the look of the website without the interactive effects and all.
- JavaScript manipulates the content to create dynamic web pages whereas HTML pages are static which means the content cannot be changed.
- JavaScript is not cross-browser compatible whereas HTML is cross-browser compatible.
- JavaScript can be embedded inside HTML but HTML can not be embedded inside JavaScript.

JavaScript Used for

It is mainly used to develop websites and web-based applications. JavaScript is a language that can be used as a front-end as well as a backend.

- **Creating Interactive Websites:** JavaScript is used to make web pages dynamic and interactive. It means using JavaScript, we can change the web page content and styles dynamically.
- **Building Applications:** JavaScript is used to make web and mobile applications. To build web and mobile apps, we can use the most popular JavaScript frameworks like – ReactJS, React Native, Node.js etc.
- **Web Servers:** We can make robust server applications using JavaScript. To be precise we use JavaScript frameworks like Node.js and Express.js to build these servers.
- **Game Development:** JavaScript can be used to design Browser games. In JavaScript, lots of game engines are available that provide frameworks for building games.

Things that makes JavaScript demanding

JavaScript is the most popular and hence the most loved language around the globe. Apart from this, there are abundant reasons to become the most demanding. Below are a listing of a few important points:

- **No need for compilers:** Since JavaScript is an interpreted language, therefore it does not need any compiler for compilation.
- **Used both Client and Server-side:** Earlier JavaScript was used to build client-side applications only, but with the evolution of its frameworks namely Node.js and Express.js, it is now widely used for building server-side applications too.
- **Helps to build a complete solution:** As we saw, JavaScript is widely used in both client and server-side applications, therefore it helps us to build an end-to-end solution to a given problem.
- **Used everywhere:** JavaScript is so loved because it can be used anywhere. It can be used to develop websites, games or mobile apps, etc.
- **Huge community support:** JavaScript has a huge community of users and mentors who love this language and take it’s legacy forward.

JavaScript

Places to put JavaScript code

- Between the body tag of html
- Between the head tag of html
- In .js file (external javaScript)

JavaScript can be added to your HTML file in two ways:

- Internal JavaScript
- External JavaScript
- **Internal JavaScript:** We can add JS code directly to our HTML file by writing the code inside the <script> & </script>. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
- **External JavaScript:** We can create the file with a .js extension (eg.- xyz.js) and paste the JS code inside of it. After creating the file, add this file in <script src=”xyz.js”> tag, This <script> can be imported inside <head> or <body> tag of the HTML file. It provides **code re usability** because single JavaScript file can be used in several html pages.

Internal JS Example

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```

- The **script** tag specifies that we are using JavaScript.
- The **text/javascript** is the content type that provides information to the browser about the data.
- The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

Types of JavaScript Comments

There are two types of comments in JavaScript.

- Single-line Comment

It is represented by double forward slashes (//). For example:

```
<script>
// It is single line comment
document.write(" example of javascript single line comment ");
</script>
```

- Multi-line Comment

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

JavaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.

Example

- var x = 10;
- var _a=“vit";

JavaScript Variable

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

<script>

function abc(){

var x=10;//local variable

}

</script>

Or,

<script>

If(10**<13**){

var y=20;//JavaScript local variable

}

</script>

JavaScript Variable

JavaScript global variable

A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

JavaScript Variable

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
function m(){
window.value=100;//declaring global variable by window object
}
function n(){
alert(window.value);//accessing global variable from other function
}
```

Declaring JavaScript global variable outside function

```
var value=50;
function a(){
alert(window.value);//accessing global variable
}
```

Javascript Data Types

JavaScript provides different data types to hold different types of values.

There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

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

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript Special Operators

? : (Conditional)

If Condition is true? Then value X : Otherwise value Y

JavaScript Function

What is Function?

A function is a group of statements that perform specific tasks and can be kept and maintained separately form main program. Functions provide a way to create reusable code packages which are more portable and easier to debug. Here are some advantages of using functions:

JavaScript Function

Function without parameters

Function Definition The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}, as given below:

```
function functionName() {
   // Code to be executed
}
```

Function call

Once a function is defined it can be called (invoked) from anywhere in the document, by typing its name followed by a set of parentheses as given below:

```
functionName();
```

Example

```
function sayHello() {
document.write(“Hello”);
}
sayHello();
```

JavaScript Function

Function with parameters

Function Definition

Parameters are set on the first line of the function inside the set of parentheses, like this:

function functionName(*parameter1*, *parameter2*, *parameter3*) {
  // Code to be executed
}

Function call

functionName(*parameter1*, *parameter2*, *parameter3*);

JavaScript Function

Default Values for Function Parameters

you can specify default values to the function parameters. This means that if no arguments are provided to function when it is called these default parameters values will be used.

Example

```
function sayHello(name = 'Guest') {
    alert('Hello, ' + name);
}
sayHello(); // 0utputs: Hello, Guest
sayHello('John'); // 0utputs: Hello, John
```

JavaScript Function

Returning Values from a Function

A function can return a value back to the script that called the function as a result using the return statement. The value may be of any type, including arrays and objects. The return statement usually placed as the last line of the function before the closing curly bracket and ends it with a semicolon, as shown in the following example.

Example

```
function getSum(num1, num2) {
    var total = num1 + num2;
    return total;
}
alert(getSum(6, 20)); // 0utputs: 26
alert(getSum(-5, 17)); // 0utputs: 12
```

JavaScript Loops

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. JavaScript now supports five different types of loops:

while — loops through a block of code as long as the condition specified evaluates to true.

do…while — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.

for — loops through a block of code until the counter reaches a specified number.

for…in — loops through the properties of an object.

for…of — loops over iterable objects such as arrays, strings, etc.

JavaScript Loops

The while Loop

This is the simplest looping statement provided by JavaScript.

The while loop loops through a block of code as long as the specified condition evaluates to true. As soon as the condition fails, the loop is stopped. The generic syntax of the while loop is:

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

Example

```
var i = 1;
while(i <= 5);
{
   document.write( "<p>The number is " + i +  "</p>");
   i++;
}
```

JavaScript Loops

The do...while Loop

The do-while loop is a variant of the while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true. The generic syntax of the do-while loop is:

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

JavaScript Loops

Difference Between while and do...while Loop

The while loop differs from the do-while loop in one important way — with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a do-while loop, on the other hand, the loop will always be executed once even if the conditional expression evaluates to false, because unlike the while loop, the condition is evaluated at the end of the loop iteration rather than the beginning.

JavaScript Loops

The for Loop

The for loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times. Its syntax is:

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

The parameters of the for loop statement have following meanings:

initialization — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.

condition — it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.

increment — it updates the loop counter with a new value each time the loop runs.

JavaScript Loops

The for...in Loop

The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array. The generic syntax of the for-in loop is:

```
for(variable in object) {
   // Code to be executed
}
```

The loop counter i.e. *variable* in the for-in loop is a string, not a number. It contains the name of current property or the index of the current array element.

JavaScript Loops

for...of Loop

for-of loop which allows us to iterate over arrays or other iterable objects (e.g. strings) very easily. Also, the code inside the loop is executed for each element of the iterable object.

The following example will show you how to loop through arrays and strings using this loop.

Example

```
// Iterating over array
let letters = ["a", "b", "c", "d", "e", "f"];
for(let letter of letters) {
    console.log(letter); // a,b,c,d,e,f
}

// Iterating over string
let greet = "Hello World!";
for(let character of greet) {
    console.log(character); // H,e,l,l,o, ,W,o,r,l,d,!
}
```

JavaScript Arrays

Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. JavaScript arrays can store any valid value, including strings, numbers, objects, functions, and even other arrays, thus making it possible to create more complex data structures such as an array of objects or an array of arrays.

Creating an Array

The simplest way to create an array in JavaScript is enclosing a comma-separated list of values in square brackets ([]), as shown in the following syntax:

```
var myArray = [element0, element1, ..., elementN];
```

Array can also be created using the Array() constructor as shown in the following syntax. However, for the sake of simplicity previous syntax is recommended.

```
var myArray = new Array(element0, element1, ..., elementN);
```

JavaScript Arrays

Accessing the Elements of an Array

Array elements can be accessed by their index using the square bracket notation. An index is a number that represents an element's position in an array.

Array indexes are zero-based. This means that the first item of an array is stored at index 0, not 1, the second item is stored at index 1, and so on. Array indexes start at 0 and go up to the number of elements minus 1. So, array of five elements would have indexes from 0 to 4.

The following example will show you how to get individual array element by their index.

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

document.write(fruits[0]); // Prints: Apple

document.write(fruits[1]); // Prints: Banana

document.write(fruits[2]); // Prints: Mango

document.write(fruits[fruits.length - 1]); // Prints: Papaya
```

JavaScript Arrays

Getting the Length of an Array

The length property returns the length of an array, which is the total number of elements contained in the array. Array length is always greater than the index of any of its element.

Example

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits.length); // Prints: 5
```

Looping Through Array Elements

You can use **for loop** to access each element of an array in sequential order, like this:

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
for(var i = 0; i < fruits.length; i++) {
    document.write(fruits[i] + "<br>"); // Print array element
}
```

We can also iterate over the array elements using **for-in loop**, like this:

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
for(var i in fruits) {
    document.write(fruits[i] + "<br>");
}
```

JavaScript Arrays

Adding New Elements to an Array

To add a new element at the end of an array, simply use the push() method, like this:

Example Try this code »

```
var colors = ["Red", "Green", "Blue"];
colors.push("Yellow");

document.write(colors); // Prints: Red,Green,Blue,Yellow
document.write(colors.length); // Prints: 4
```

JavaScript Arrays

Removing Elements from an Array

To remove the last element from an array you can use the **pop() method**. This method returns the value that was popped out. Here's an example:

Example

```
var colors = ["Red", "Green", "Blue"];
var last = colors.pop();

document.write(last); // Prints: Blue
document.write(colors.length); // Prints: 2
```

Similarly, you can remove the first element from an array using **the shift() method**. This method also returns the value that was shifted out. Here's an example:

Example

```
var colors = ["Red", "Green", "Blue"];
var first = colors.shift();
document.write(first); // Prints: Red
document.write(colors.length); // Prints: 2
```

JavaScript Arrays

Adding or Removing Elements at Any Position

The splice() method is a very versatile array method that allows you to add or remove elements from any index, using the
syntax
arr.splice(startIndex, deleteCount, elem1, ..., elemN).

This method takes three parameters: the first parameter is the index at which to start splicing the array, it is required; the second parameter is the number of elements to remove (use 0 if you don't want to remove any elements), it is optional; and the third parameter is a set of replacement elements, it is also optional.

JavaScript Arrays

Creating a String from an Array

There may be situations where you simply want to create a string by joining the elements of an array. To do this you can use the join() method. This method takes an optional parameter which is a separator string that is added in between each element. If you omit the separator, then JavaScript will use comma (,) by default. The following example shows how it works:

Example

```
var colors = ["Red", "Green", "Blue"];
 document.write(colors.join()); // Prints: Red,Green,Blue
document.write(colors.join("")); // Prints: RedGreenBlue
document.write(colors.join("-")); // Prints: Red-Green-Blue
document.write(colors.join(", ")); // Prints: Red, Green, Blue
```

You can also convert an array to a comma-separated string using the toString(). This method does not accept the separator parameter like join().

Example

```
var colors = ["Red", "Green", "Blue"];
document.write(colors.toString()); // Prints: Red,Green,Blue
```

JavaScript Arrays

Extracting a Portion of an Array

If you want to extract out a portion of an array (i.e. subarray) but keep the original array intact you can use the slice() method. This method takes 2 parameters: start index (index at which to begin extraction), and an optional end index (index before which to end extraction), like arr.slice(startIndex, endIndex). Here's an example:

Example

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
var subarr = fruits.slice(1, 3);
document.write(subarr); // Prints: Banana,Mango
```

If endIndex parameter is omitted, all elements to the end of the array are extracted. You can also specify negative indexes or offsets —in that case the slice() method extract the elements from the end of an array, rather then the begining.

For example:

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits.slice(2)); // Prints: Mango,Orange,Papaya
document.write(fruits.slice(-2)); // Prints: Orange,Papaya
document.write(fruits.slice(2, -1)); // Prints: Mango,Orange
```

JavaScript Arrays

Merging Two or More Arrays

The concat() method can be used to merge or combine two or more arrays. This method does not change the existing arrays, instead it returns a new array. For example:

Example

```
var pets = ["Cat", "Dog", "Parrot"];
var wilds = ["Tiger", "Wolf", "Zebra"];
var bugs = ["Ant", "Bee"];

// Creating new array by combining pets, wilds and bugs arrays
var animals = pets.concat(wilds, bugs);
document.write(animals); // Prints: Cat,Dog,Parrot,Tiger,Wolf,Zebra,Ant,Bee
```

The concat() method can take any number of array arguments, so you can create an array from any number of other arrays,

JavaScript Arrays

Searching Through an Array

If you want to search an array for a specific value, you can simply use the indexOf() and lastIndexOf().

- If the value is found, both methods return an index representing the array element.
- If the value is not found, -1 is returned.
- The indexOf() method returns the first one found, whereas
- the lastIndexOf() returns the last one found.
- First argument of both the methods is an array element to be searched and the second argument (optional) specifies the index within the array at which to start the search.

Example

Index 0 1 2 3 4 5 6 7 8

```
var arr = [1, 0, 3, 1, 2, 5, 1, 4, 7];
document.write(arr.indexOf(1, 2)); // Searching forwards Prints: 3
document.write(arr.lastIndexOf(1, 2)); // Searching backwards, Prints: 0
```

JavaScript Arrays

Searching Through an Array

We can also use **includes() method** to find out whether an array includes a certain element or not. This method takes the same parameters as indexOf() and lastIndexOf() methods, but it returns true or false instead of index number. For example:

Example

```
var arr = [1, 0, 3, 1, 2, 5, 1, 4, 7];
document.write(arr.includes(1)); // Prints: true
document.write(arr.includes(6)); // Prints: false
document.write(arr.includes(1, 2)); // Prints: true
document.write(arr.includes(3, 4)); // Prints: false
```

JavaScript Arrays

Searching Through an Array

If you want to search an array based on certain condition then you can use the **JavaScript find() method**. This method returns the value of the first element in the array that satisfies the provided testing function. Otherwise it returns undefined.

Example

```
var arr = [1, 0, 3, 1, 2, 5, 1, 4, 7];

var result = arr.find(function(element) {
  return element > 4;
});
var result2 = arr.find(function(element){
        return element > 7;
 });
document.write(result+"<br>"); // Prints: 5
document.write(result2); // Prints: undefined
```

JavaScript Arrays

Searching Through an Array

There is one more method similar to find() is **findIndex() method**, which returns the index of a found element in the array instead of its value. For example:

Example

```
var arr = [1, 0, 3, 1, 2, 5, 1, 4, 7];

var result = arr.findIndex(function(element) {
  return element > 6;
});
document.write(result); // Prints: 8
```

JavaScript Arrays

Searching Through an Array

If you want to find out all the matched elements you can use **the filter() method**. The filter() method creates a new array with all the elements that successfully passes the given test. The following example will show you how this actually works:

Example

```
var arr = [1, 0, 3, 1, 2, 5, 1, 4, 7];

var result = arr.filter(function(element) {
  return element > 4;
});
document.write(result); // Prints: 5,7
document.write(result.length); // Prints: 2
```

JavaScript Arrays

Sorting string Array

Sorting is a common task when working with arrays. It would be used, for instance, if you want to display the city or county names in alphabetical order.

The JavaScript Array object has a built-in method sort() for sorting array elements in alphabetical order. The following example demonstrates how it works:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Papaya", "Mango"];
fruits.sort();
document.write(fruits ); // Outputs: Apple,Banana,Mango,Orange,Papaya
```

Sorting Numeric Arrays

The sort() method may produce unexpected result when it is applied on the numeric arrays (i.e. arrays containing numeric values). For instance:

Example

```
var numbers = [5, 20, 10, 75, 50, 100];
numbers.sort(); // Sorts numbers array
document.write(numbers); // Outputs: 10,100,20,5,50,75
```

JavaScript Arrays

Sorting Numeric Arrays

It happens because, the sort() method sorts the numeric array elements by converting them to strings (i.e. 20 becomes "20", 100 becomes "100", and so on), and since the first character of string "20" (i.e. "2") comes after the first character of string "100" (i.e. "1"), that's why the value 20 is sorted after the 100.

To fix this sorting problem with numeric array, you can pass a compare function, like this:

Example

```
var numbers = [5, 20, 10, 75, 50, 100];
// Sorting an array using compare function
    numbers.sort(function(a, b) {
    return a - b;
});
document.write(numbers); // Outputs: 5,10,20,50,75,100
```

JavaScript Arrays

Sorting Numeric Arrays

when comparing a and b:

- If the compare function returns a value less than 0, then a comes first.
- If the compare function returns a value greater than 0, then b comes first.
- If the compare function returns 0, a and b remain unchanged with respect to each other, but sorted with respect to all other elements.

Hence, since 5 - 20 = -15 which is less than 0, therefore 5 comes first, similarly 20 - 10 = 10 which is greater than 0, therefore 10 comes before 20, likewise 20 - 75 = -55 which is less than 0, so 20 comes before 75, similarly 50 comes before 75, and so on.

JavaScript Arrays

Reversing an Array

We can use the reverse() method to reverse the order of the elements of an array.
This method reverses an array in such a way that the first array element becomes the last, and the last array element becomes the first. Here's an example:

Example

```
var counts = ["one", "two", "three", "four", "five"];
var reversed = counts.reverse();
document.write(counts + "<br>"); // Outputs: five,four,three,two,one
document.write(reversed); // Output: five,four,three,two,one
```

Note: The sort() and reverse() method modifies the original array and return a reference to the same array, as you can see in the above examples.

JavaScript Arrays

Finding the Maximum and Minimum Value in an Array

You can use the apply() method in combination with the Math.max() and Math.min() to find the maximum and minimum value inside an array, like this:

Example

```
var numbers = [3, -7, 10, 8, 15, 2];
// Defining function to find maximum value
function findMax(array) {
    return Math.max.apply(null, array);
}
// Defining function to find minimum value
function findMin(array) {
    return Math.min.apply(null, array);
}
document.write(findMax(numbers) + "<br>"); // Outputs: 15
document.write(findMin(numbers)); // Outputs: -7
```

JavaScript Arrays

Finding the Maximum and Minimum Value in an Array

The apply() method provides a convenient way to pass array values as arguments to a function that accepts multiple arguments in an array-like manner, but not an array (e.g. Math.max() and Math.min() methods here). So, the resulting statement Math.max.apply(null, numbers) in the example above is equivalent to the Math.max(3, -7, 10, 8, 15, 2).

JavaScript Arrays

Sorting an Array of Objects

The sort() method can also be used for sorting object arrays using the compare function.

The following example will show you how to sort an array of objects by property values:

THANK YOU