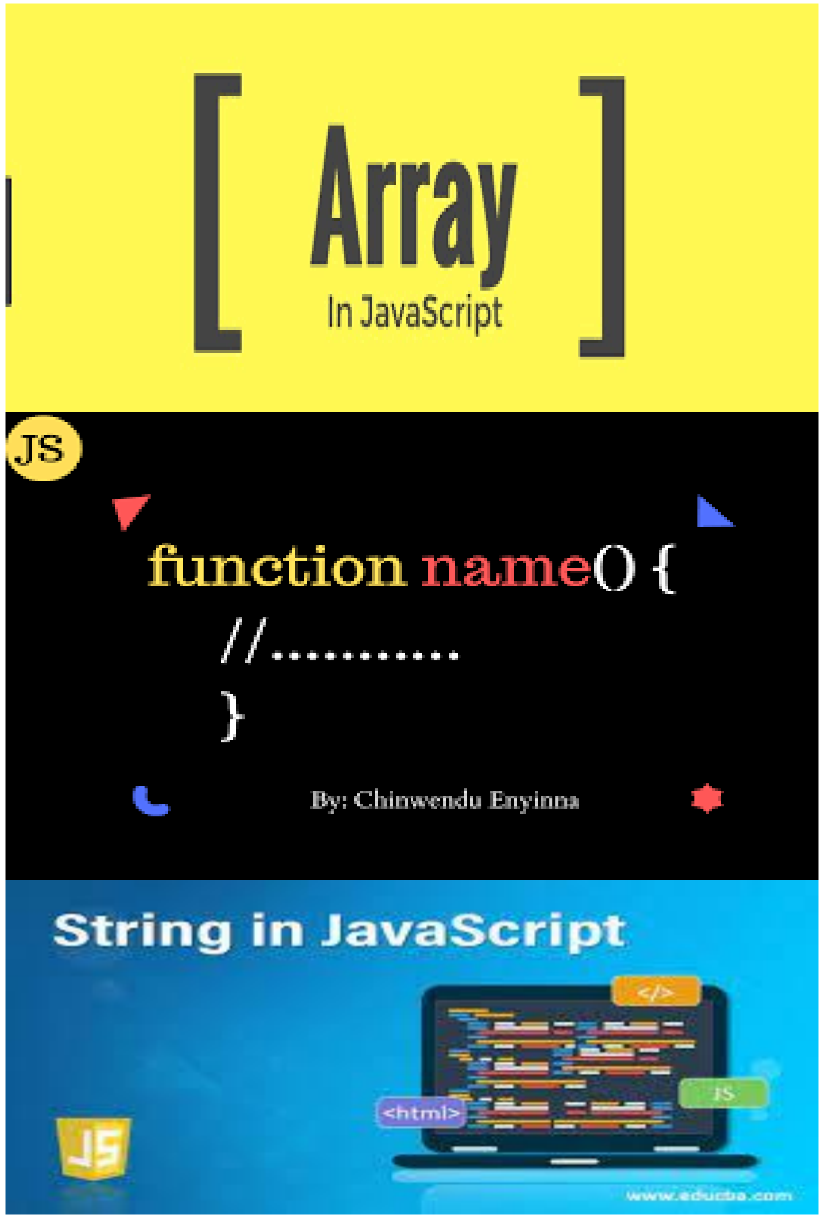


Chapter 2
Array, Function and String

Arrays in JavaScript:

- JavaScript array is a single variable that is used to store different elements. It is often used when we want to store a list of elements and access them by a single variable.

- In JavaScript, an array is a single variable that stores multiple elements.

- An array is a special variable, which can hold more than one value:

const cars = ["Saab", "Volvo", "BMW"];

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

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

Declaration of an Array: There are basically two ways to declare an array.

```
const arrayName = [value1, value2, ...]; // Method 1

const arrayName = new Array(value1, value2, ...);// Method 2
```

Example:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

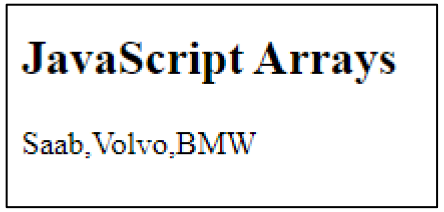
You can also create an array, and then provide the elements:

Example:

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Example:

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```



```
JavaScript Arrays

Saab,Volvo,BMW
```

Using the JavaScript Keyword new:

The following example also creates an Array, and assigns values to it:

const cars = new Array("Saab", "Volvo", "BMW");

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

```
JavaScript Arrays
Saab,Volvo,BMW
```

For simplicity, readability and execution speed, use the array literal method.

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
```

```
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes
(starting from 0).</p>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

OUTPUT:

```
JavaScript Arrays
JavaScript array elements are accessed using
numeric indexes (starting from 0).

Saab
```

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

OUTPUT

```
JavaScript Arrays
JavaScript array elements are accessed using numeric indexes (starting from 0).
Opel,Volvo,BMW
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

<u>OUTPUT</u>

```
JavaScript Arrays

Saab,Volvo,BMW
```

Arrays are Objects

- Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.
- But, JavaScript arrays are best described as arrays.
- Arrays use numbers to access its "elements". In this example, person[0] returns John:

```
<script>
const person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
</script>
```

```
John
```

Objects use names to access its "members". In this example, person.firstName returns John:

```
<script>
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML =
person.firstName;
</script>
```

```
John
```

Array Elements Can Be Objects

- Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
<html>
<body>

<h2>JavaScript Arrays</h2>
<p>The length property returns the length of an array.</p>

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

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

</body>
</html>
```

OUTPUT

```
JavaScript Arrays
The length property returns the length of an array.

4
```

Note:The length property is always one more than the highest array index.

Accessing the First Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

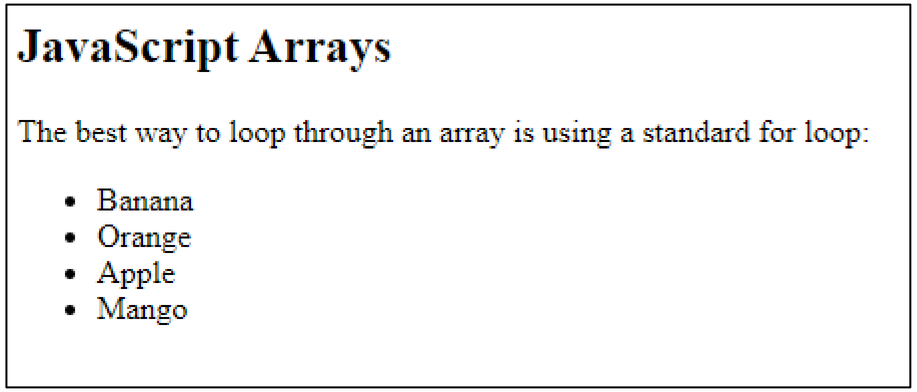
Accessing the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Looping Array Elements

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for loop:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;
let text = "<ul>";
for (let i = 0; i < fLen; i++)
{
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



```
JavaScript Arrays

The best way to loop through an array is using a standard for loop:

  • Banana
  • Orange
  • Apple
  • Mango
```

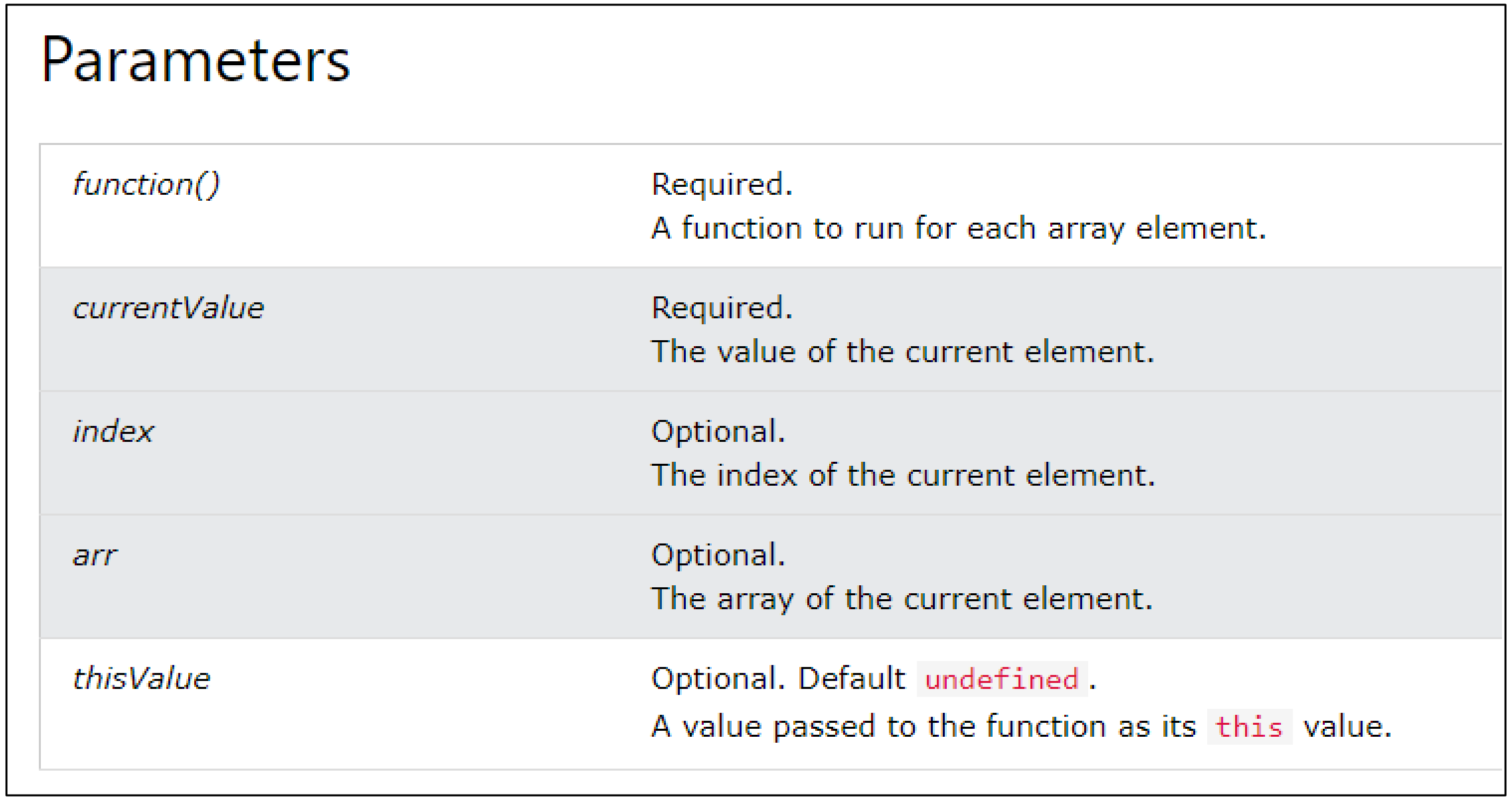
JavaScript Array forEach()

Definition and Usage

- The forEach() method calls a function for each element in an array.
- The forEach() method is not executed for empty elements.

Syntax

```
array.forEach(function(currentValue, index, arr), thisValue)
```



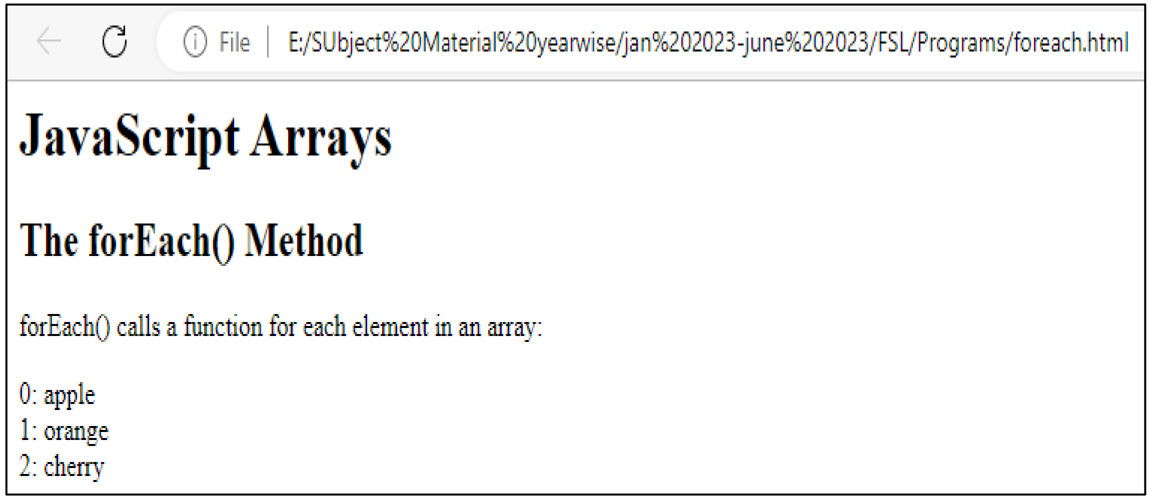
Parameters

function()	Required. A function to run for each array element.
currentValue	Required. The value of the current element.
index	Optional. The index of the current element.
arr	Optional. The array of the current element.
thisValue	Optional. Default `undefined`. A value passed to the function as its `this` value.

JavaScript Array forEach()

Calls a function for each element in fruits:

```
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p id="demo"></p>
<script>
let text = "";
const fruits = ["apple", "orange", "cherry"];
fruits.forEach(myFunction);
document.getElementById("demo").innerHTML = text;
function myFunction(item, index)
{
  text += index + ": " + item + "<br>";
}
</script>
</body>
</html>
```


File
E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/foreach.html

JavaScript Arrays

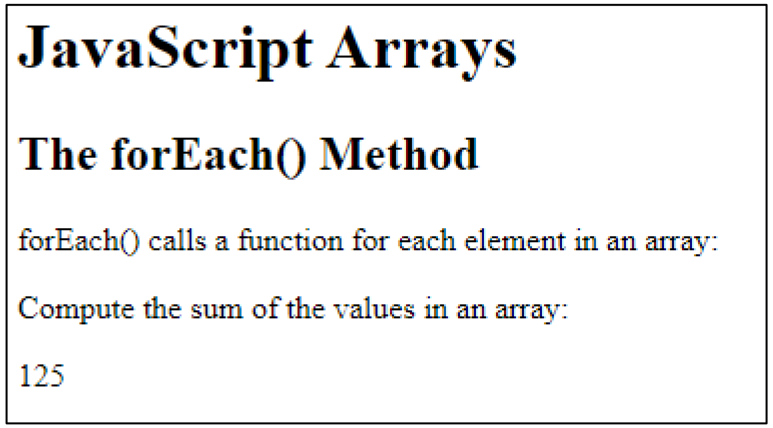
The forEach() Method

forEach() calls a function for each element in an array:

```
0: apple
1: orange
2: cherry
```

Compute the sum: using FOR-EACH

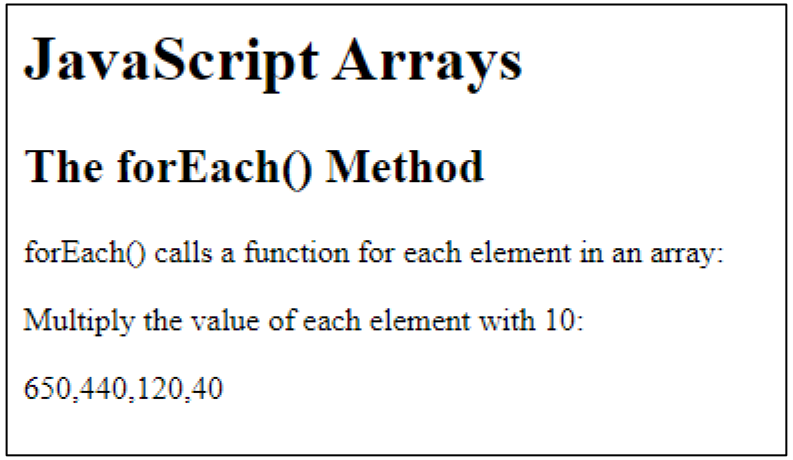
```
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p>Compute the sum of the values in an array:</p>
<p id="demo"></p>
<script>
let sum = 0;
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = sum;
function myFunction(item)
{
  sum += item;
}
</script>
</body>
</html>
```



```
JavaScript Arrays

The forEach() Method

forEach() calls a function for each element in an array:

Compute the sum of the values in an array:

125
```

Multiply each element:

```
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p>Multiply the value of each element with 10:</p>
<p id="demo"></p>
<script>
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction)
document.getElementById("demo").innerHTML = numbers;
function myFunction(item, index, arr) {
  arr[index] = item * 10;
}
</script>
</body>
</html>
```



JavaScript Arrays

The forEach() Method

forEach() calls a function for each element in an array:

Multiply the value of each element with 10:

```
650,440,120,40
```

JavaScript for...in Loop

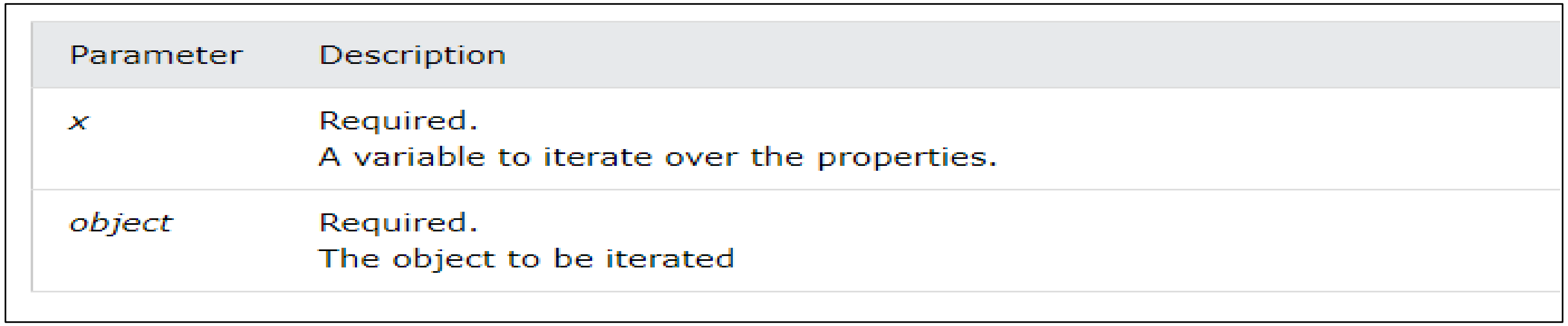
Definition and Usage

- The for...in statements combo iterates (loops) over the properties of an object.
- The code block inside the loop is executed once for each property.

Syntax

```
for (x in object)
{
  code block to be executed
}
```

Parameters



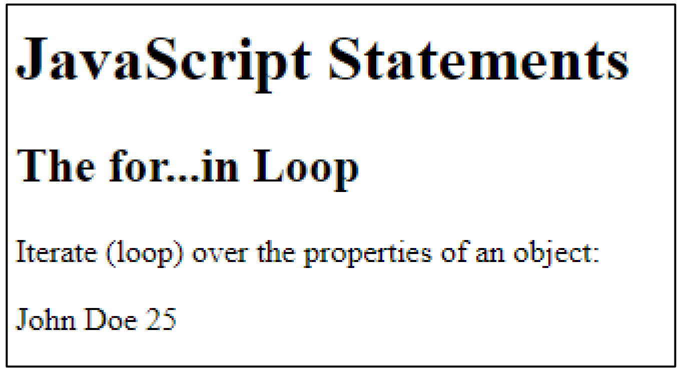
Parameter	Description
x	Required. A variable to iterate over the properties.
object	Required. The object to be iterated

Examples

Iterate (loop) over the properties of an object:

```
<html>
<body>
<h1>JavaScript Statements</h1>
<h2>The for...in Loop</h2>
<p>Iterate (loop) over the properties of an object:</p>
<p id="demo"></p>
<script>
const person = {fname:"John", lname:"Doe", age:25};
let text = "";
for (let x in person) {
  text += person[x] + " ";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

OUTPUT



JavaScript Statements

The for...in Loop

Iterate (loop) over the properties of an object:

John Doe 25

Example Explained

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

For In Over Arrays

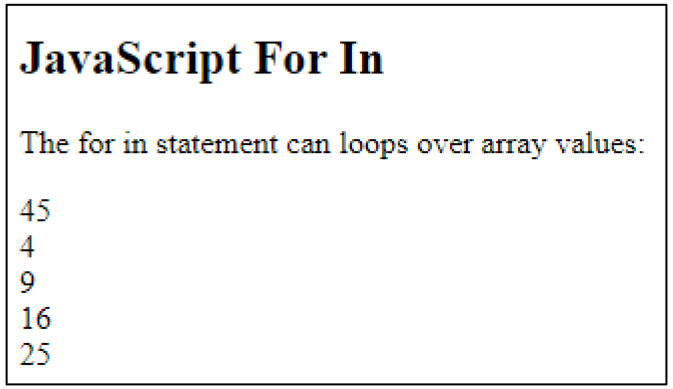
The JavaScript for in statement can also loop over the properties of an Array:

Syntax

```
for (variable in array)
{
  code
}
```

Example

```
<h2>JavaScript For In</h2>
<p>The for in statement can loops over array values:</p>
<p id="demo"></p>
<script>
const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
  txt += numbers[x] + "<br>";
}
document.getElementById("demo").innerHTML = txt;
</script>
```



JavaScript For In

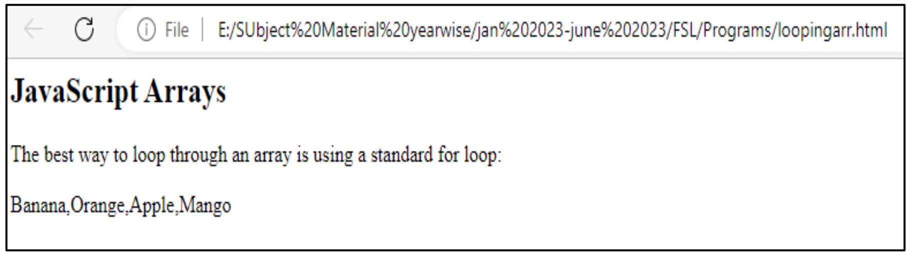
The for in statement can loops over array values:

```
45
4
9
16
25
```

Looping Array Elements

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for
loop:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

for (let i = 0; i < fLen; i++)
{
  fruits[i];
}

document.getElementById("demo").innerHTML = fruits;
</script>
</body>
</html>
```



File | E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/loopingarr.html

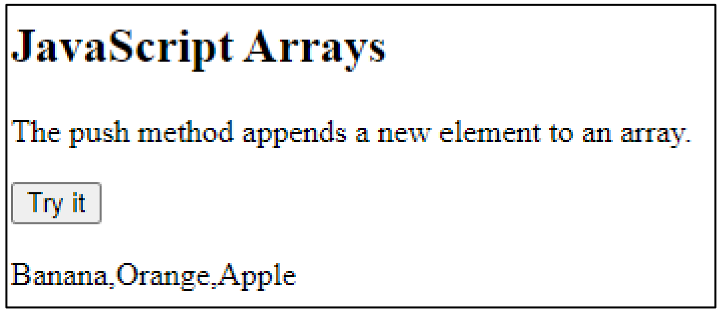
```
JavaScript Arrays

The best way to loop through an array is using a standard for loop:

Banana,Orange,Apple,Mango
```

Adding Array Elements

To add a new element to an array is using the push() method:

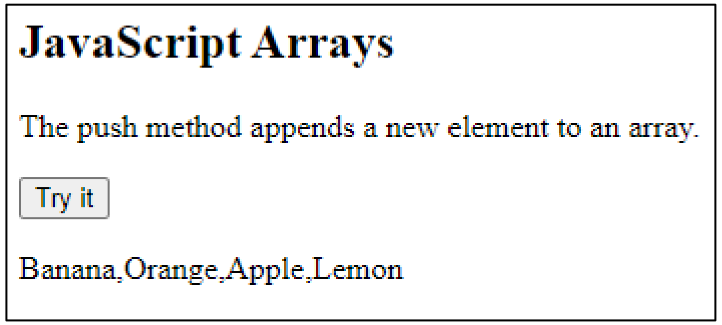
Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The push method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>
</body>
</html>
```


JavaScript Arrays

The push method appends a new element to an array.

```
Banana,Orange,Apple
```

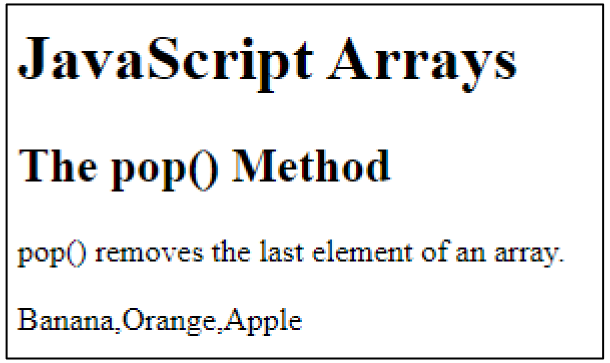

JavaScript Arrays

The push method appends a new element to an array.

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

Removing an Array Elements

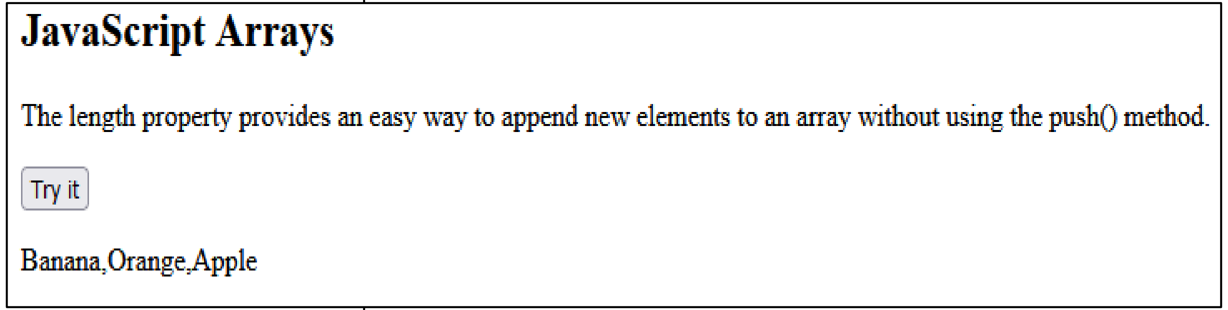
```
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The pop() Method</h2>

<p>pop() removes the last element of an array.</p>

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

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

</body>
</html>
```



```
JavaScript Arrays

The pop() Method

pop() removes the last element of an array.

Banana,Orange,Apple
```

New element can also be added to an array using the length property:

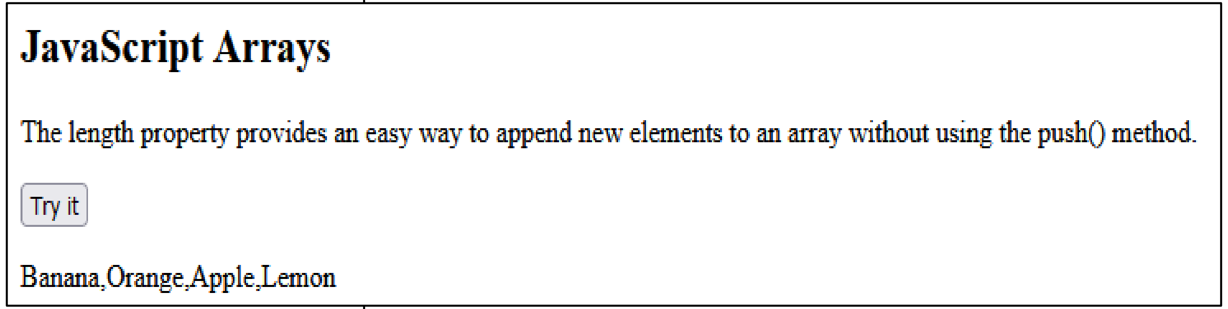
```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The length property provides an easy way to append new elements to an
array without using the push() method.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits[fruits.length] = "Lemon";
  document.getElementById("demo").innerHTML = fruits;
}
</script>
</body>
</html>
```



JavaScript Arrays

The length property provides an easy way to append new elements to an array without using the push() method.

```
Banana,Orange,Apple
```



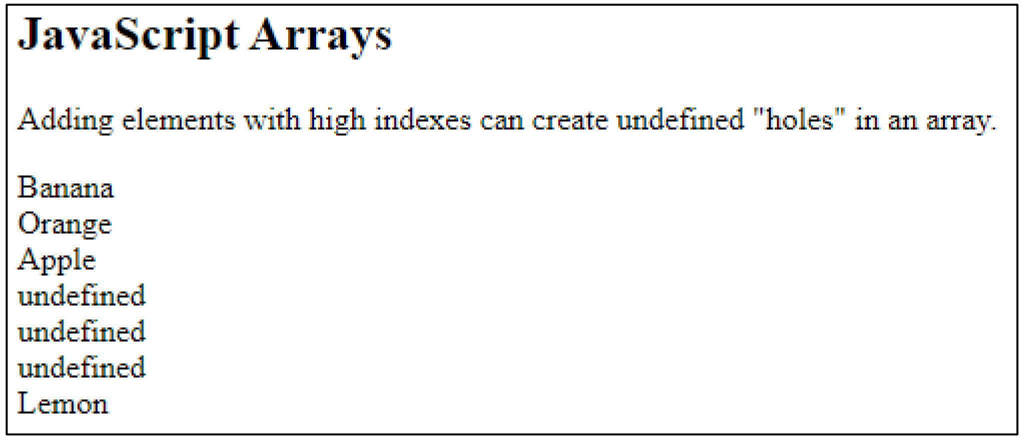
JavaScript Arrays

The length property provides an easy way to append new elements to an array without using the push() method.

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

Note:Adding elements with high indexes can create undefined "holes" in an array:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>Adding elements with high indexes can create undefined "holes" in an array.</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";
let fLen = fruits.length;
let text = "";
for (i = 0; i < fLen; i++) {
  text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



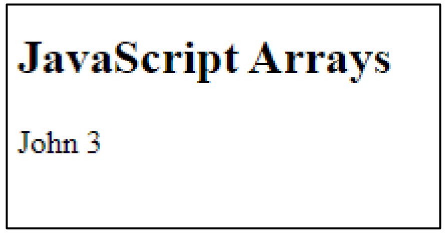
```
JavaScript Arrays

Adding elements with high indexes can create undefined "holes" in an array.

Banana
Orange
Apple
undefined
undefined
undefined
Lemon
```

Associative Arrays

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does not support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
</body>
</html>
```

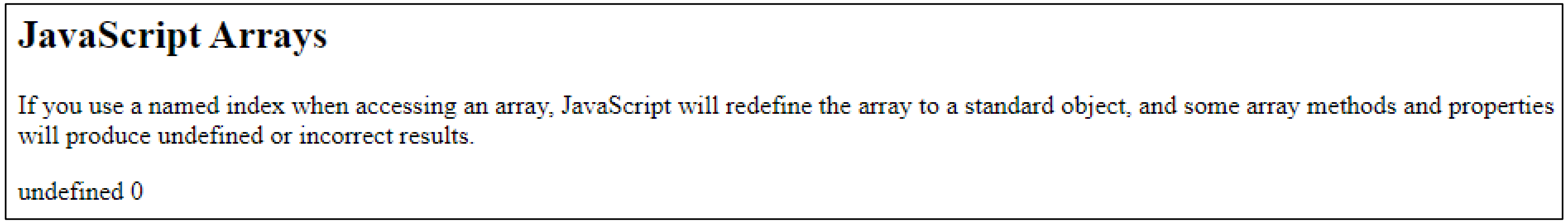


```
JavaScript Arrays

John 3
```

WARNING !!

- **If you use named indexes, JavaScript will redefine the array to an object.**
- **After that, some array methods and properties will produce incorrect results.**

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some
array methods and properties will produce undefined or incorrect results.</p>
<p id="demo"></p>
<script>
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
</body>
</html>
```



JavaScript Arrays

If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some array methods and properties will produce undefined or incorrect results.

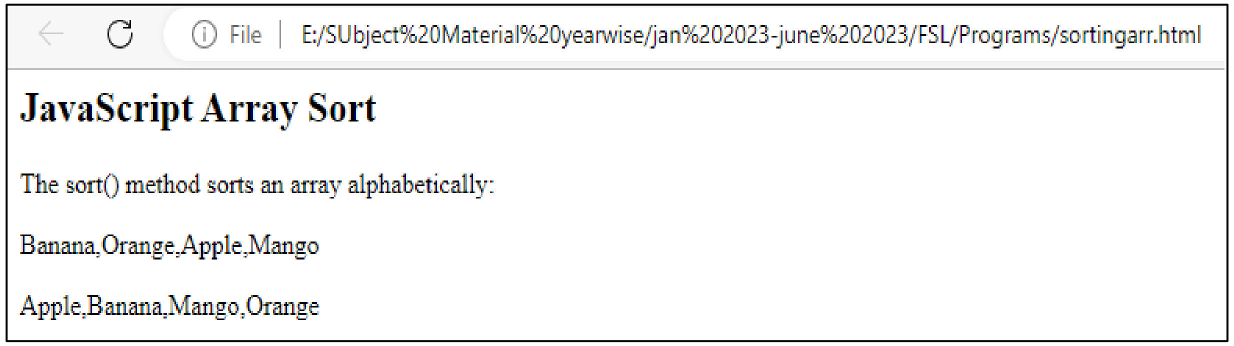
```
undefined 0
```

JavaScript Sorting Arrays

The sort() method sorts an array alphabetically:

Example

```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>The sort() method sorts an array alphabetically:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.sort();
document.getElementById("demo2").innerHTML = fruits;
</script>
</body>
</html>
```


File
E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/sortingarr.html

JavaScript Array Sort

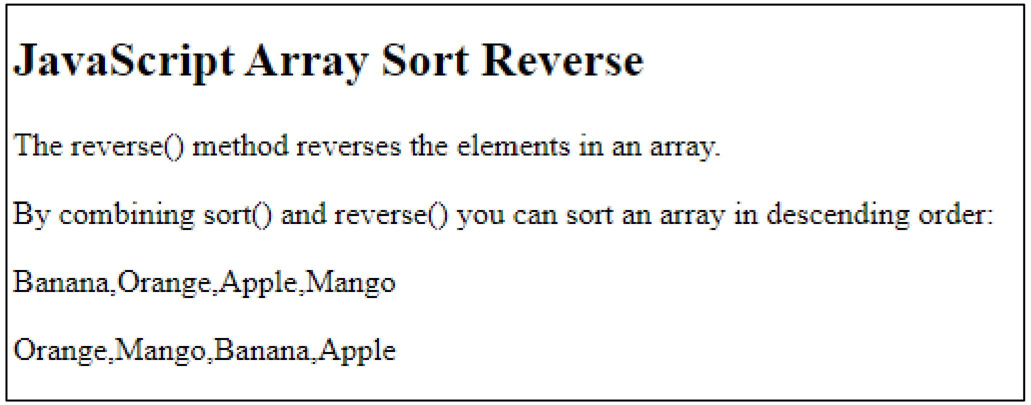
The sort() method sorts an array alphabetically:

```
Banana,Orange,Apple,Mango

Apple,Banana,Mango,Orange
```

Reversing an Array

- The reverse() method reverses the elements in an array.
- You can use it to sort an array in descending order:

```
<html>
<body>
<h2>JavaScript Array Sort Reverse</h2>
<p>The reverse() method reverses the elements in an array.</p>
<p>By combining sort() and reverse() you can sort an array in descending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
// Create and display an array:
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
// Then reverse it:
fruits.reverse();
document.getElementById("demo2").innerHTML = fruits;
</script>
</body>
</html>
```



JavaScript Array Sort Reverse

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

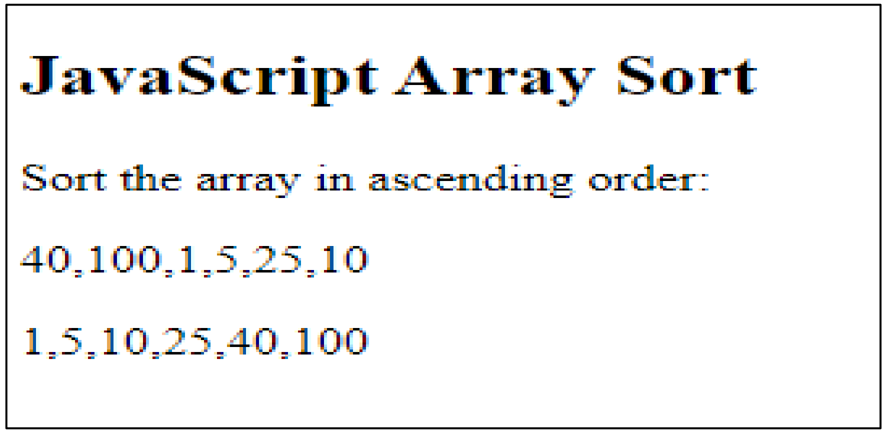
By combining sort() and reverse() you can sort an array in descending order:

```
Banana,Orange,Apple,Mango

Orange,Mango,Banana,Apple
```

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.
- You can fix this by providing a compare function:

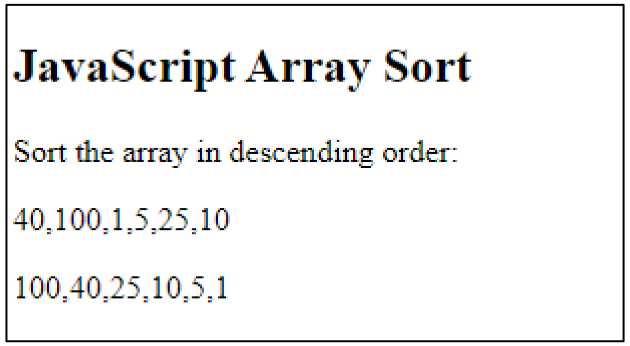
```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>Sort the array in ascending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = points;
points.sort(function(a, b){return a - b});
document.getElementById("demo2").innerHTML = points;
</script>
</body>
</html>
```



```
JavaScript Array Sort

Sort the array in ascending order:

40,100,1,5,25,10

1,5,10,25,40,100
```

Sort an array descending

```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>Sort the array in descending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = points;
points.sort(function(a, b){return b - a});
document.getElementById("demo2").innerHTML = points;
</script>
</body>
</html>
```



```
JavaScript Array Sort

Sort the array in descending order:

40,100,1,5,25,10

100,40,25,10,5,1
```

The Compare Function

- The purpose of the compare function is to define an alternative sort order.
- The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a - b}
```

- When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

- If the result is negative, a is sorted before b.

- If the result is positive, b is sorted before a.

- If the result is 0, no changes are done with the sort order of the two values.

Example:

- The compare function compares all the values in the array, two values at a time (a, b).

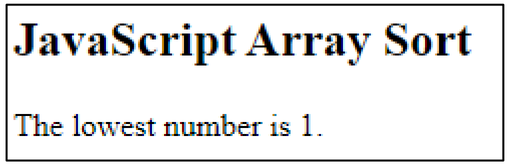
- When comparing 40 and 100, the sort() method calls the compare function(40, 100).

- The function calculates 40 - 100 (a - b), and since the result is negative (-60), the sort function will sort 40 as a value lower than 100.

Find the Highest (or Lowest) Array Value

Lowest

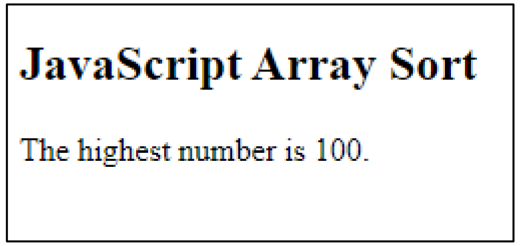
```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>The lowest number is <span id="demo"></span>.</p>
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
document.getElementById("demo").innerHTML = points[0];
</script>
</body>
</html>
```



```
JavaScript Array Sort

The lowest number is 1.
```

Highest

```
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b-a});
document.getElementById("demo").innerHTML = points[0];
</script>
```



```
JavaScript Array Sort

The highest number is 100.
```

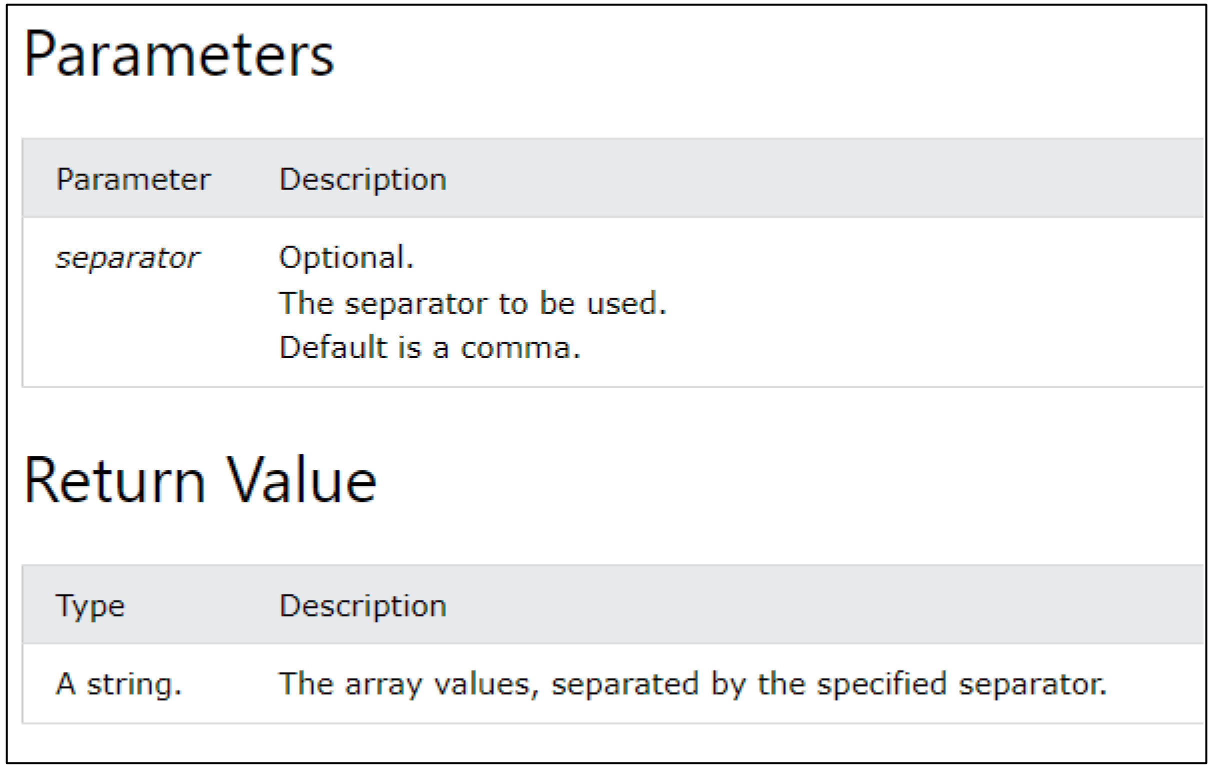
Combining an Array element into a String

JavaScript Array join()

- The join() method returns an array as a string.
- The join() method does not change the original array.
- Any separator can be specified. The default is comma (,).

Syntax

```
array.join(separator)
```



Parameters

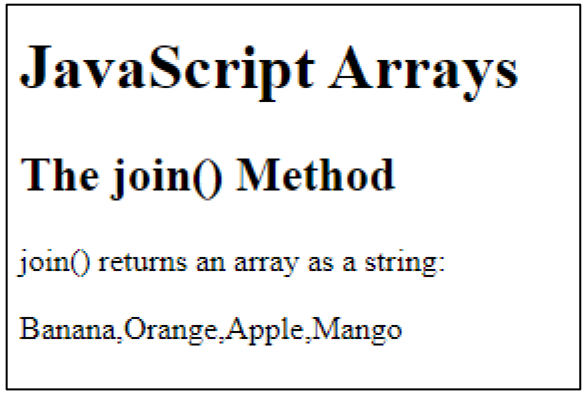
Parameter	Description
separator	Optional. The separator to be used. Default is a comma.

Return Value

Type	Description
A string.	The array values, separated by the specified separator.

Example:----join

```
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The join() Method</h2>

<p>join() returns an array as a string:</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let text = fruits.join();

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

</body>
</html>
```



JavaScript Arrays

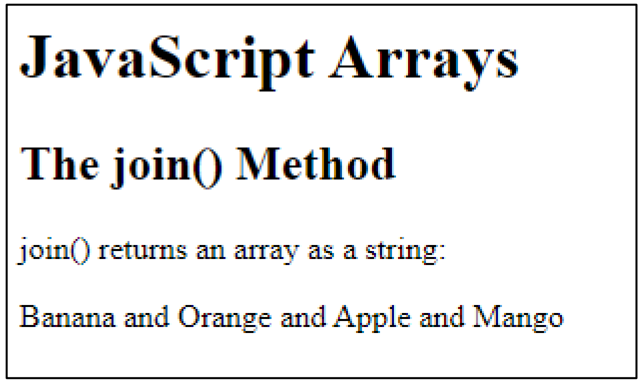
The join() Method

join() returns an array as a string:

Banana,Orange,Apple,Mango

Example:----join

```
<body>
<h1>JavaScript Arrays</h1>
<h2>The join() Method</h2>
<p>join() returns an array as a string:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let text = fruits.join(" and ");
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



JavaScript Arrays

The join() Method

join() returns an array as a string:

Banana and Orange and Apple and Mango

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)

- The code to be executed, by the function, is placed inside curly brackets: {}

```
function name(parameter1, parameter2, parameter3)
{
  // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a return value. The return value is "returned" back to the "caller":

Example:

Calculate the product of two numbers, and return the result:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation
and returns the result:</p>
<p id="demo"></p>
<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>
</body>
</html>
```

Why Functions?

- **Code reusability**: We can call a function several times so it save coding.
- **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function to convert from Fahrenheit to
Celsius:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML =
toCelsius(77);
</script>
</body>
</html>
```

```
JavaScript Functions
This example calls a function to convert from
Fahrenheit to Celsius:

25
```

The () Operator Invokes the Function

Accessing a function without () will return the function object instead of the function result.

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>Accessing a function without () will return the function
definition instead of the function result:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius;
</script>
</body>
</html>
```

```
JavaScript Functions
Accessing a function without () will return the function
definition instead of the function result:

function toCelsius(f) { return (5/9) * (f-32); }
```

Local Variables

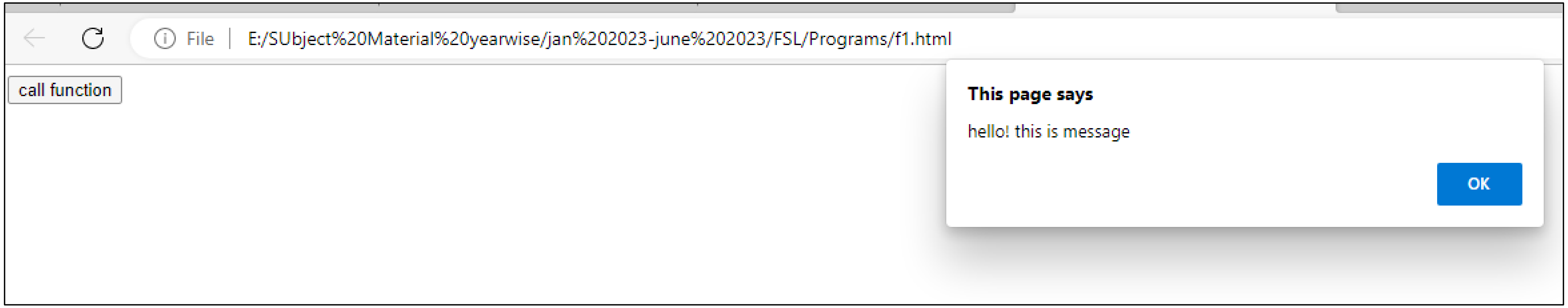
- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Function Example

```
<html>
<body>
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
</body>
</html>
```



```
File | E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/f1.html
```

This page says

hello! this is message

JavaScript Function Arguments

```
<script>
function getcube(number){
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
```

Function with Return Value

- The return statement is used to return a particular value from the function to the function caller. The function will stop executing when the return statement is called. The return statement should be the last statement in a function because the code after the return statement will be unreachable.

- We can return primitive values (such as Boolean, number, string, etc.) and Object types (such as functions, objects, arrays, etc.) by using the return statement.

- We can also return multiple values using the **return** statement. It cannot be done directly. We have to use an **Array** or **Object** to return multiple values from a function.

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
```

Example1
This is a simple example of using the return statement. Here, we are returning the result of the product of two numbers and returned back the value to the function caller.

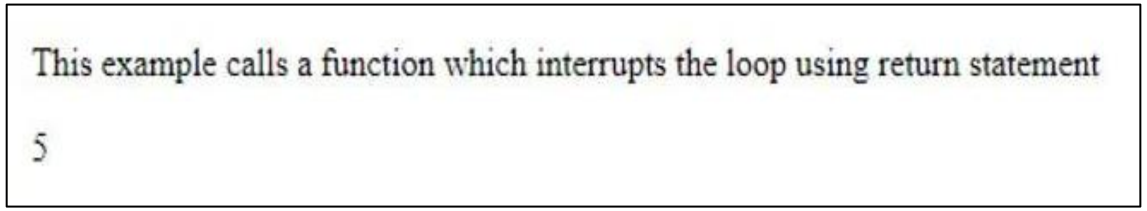
The variable res is the function caller; it is calling the function fun() and passing two integers as the arguments of the function. The result will be stored in the res variable. In the output, the value 360 is the product of arguments 12 and 30.

```
<html>
  <head>
</head>
  <body>
<h3> Example of the JavaScript's return statement </h3>
    <script>
var res = fun(12, 30);
function fun(x, y)
{
return x * y;
}
document.write(res);
    </script>
</body>
 </html>
```

```
Example of the JavaScript's return statement
360
```

Example 2

```
<html>
<body>
<p>This example calls a function which interrupts the
loop using return statement</p>
<p id="demo"></p>
<script>
function func(){
for(var i = 0; i <= 10; i++){
if(i === 5)
return i;
}
}
document.getElementById("demo").innerHTML = func();
</script>
</body>
</html>
```



```
This example calls a function which interrupts the loop using return statement

5
```

Note:

In the above program, we can see we have a function defined with the name "func" where we have created a for loop to print the value where the return statement will return the value, but in the function, it will stop at 5, and therefore the return statement will stop the for a loop at 5 though it can traverse up to 10.

Example 3

- Here, we are interrupting a function using the **return** statement. The function stops executing immediately when the **return** statement is called.
- There is an infinite **while** loop and variable **i,** which is initialized to 1. The loop continues until the value of **i** reached to **4**. When the variable's value will be 4, the loop stops its execution because of the **return** statement. The statement after the loop will never get executed.
- Here, the **return** statement is without using the ***expression***, so it returns **undefined.**

```
<script>
   var x = fun();
function fun() {
var i = 1;
  while(i) {
    document.write(i + '<br>');
      if (i == 4) {
        return;
      }
      document.write(i + '<br>');
      i++;
    }
  document.write('Hello world');
}
</script>
```

OUTPUT:

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

THANK YOU