

New Features in ES6

- The let keyword
- The const keyword
- Arrow Functions
- The ... Operator
- For/of
- Map Objects
- Set Objects
- Classes
- Promises
- Symbol
- Default Parameters
- Function Rest Parameter
- String.includes()
- String.startsWith()
- String.endsWith()
- Array.from()
- Array.keys()
- Array.find()
- Array.findIndex()
- New Math Methods
- New Number Properties
- New Number Methods
- New Global Methods
- Object entries
- JavaScript Modules

JavaScript let

The `let` keyword allows you to declare a variable with block scope.

Example

```
var x = 10;  
// Here x is 10  
{  
  let x = 2;  
  // Here x is 2  
}  
// Here x is 10
```

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

<h2>Redeclaring a Variable Using let</h2>

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

<script>
let x = 10;
// Here x is 10

{
  let x = 2;
  // Here x is 2
}

// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Redeclaring a Variable Using let

10

JavaScript const

The `const` keyword allows you to declare a constant (a JavaScript variable with a constant value).

Constants are similar to let variables, except that the value cannot be changed.

Example

```
var x = 10;  
// Here x is 10  
{  
  const x = 2;  
  // Here x is 2  
}  
// Here x is 10
```

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

<h2>Declaring a Variable Using const</h2>

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

<script>
var x = 10;
// Here x is 10
{
  const x = 2;
  // Here x is 2
}
// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Declaring a Variable Using const

10

Arrow Functions

Arrow functions allows a short syntax for writing function expressions.

You don't need the `function` keyword, the `return` keyword, and the **curly brackets**.

Example

```
// ES5
var x = function(x, y) {
  return x * y;
}

// ES6
const x = (x, y) => x * y;
```

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

<h2>JavaScript Arrow Functions</h2>

<p>With arrow functions, you don't have to type the function keyword, the return keyword,
and the curly brackets.</p>

<p>Arrow functions are not supported in IE11 or earlier.</p>

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

<script>
const x = (x, y) => x * y;
document.getElementById("demo").innerHTML = x(5, 5);
</script>

</body>
</html>
```

JavaScript Arrow Functions

With arrow functions, you don't have to type the function keyword, the return keyword, and the curly brackets.

Arrow functions are not supported in IE11 or earlier.

25

The Spread (...) Operator

The ... operator expands an iterable (like an array) into more elements:

Example

```
const q1 = ["Jan", "Feb", "Mar"];  
const q2 = ["Apr", "May", "Jun"];  
const q3 = ["Jul", "Aug", "Sep"];  
const q4 = ["Oct", "Nov", "May"];  
  
const year = [...q1, ...q2, ...q3, ...q4];
```



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

<h1>JavaScript Operators</h1>
<h2>The ... Operator</h2>

<p>The "spread" operator spreads elements of iterable objects:</p>

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

<script>
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
document.getElementById("demo").innerHTML = year;
</script>

</body>
</html>
```

JavaScript Operators

The ... Operator

The "spread" operator spreads elements of iterable objects:

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, May

The For/Of Loop

The JavaScript `for/of` statement loops through the values of an iterable objects.

`for/of` lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The `for/of` loop has the following syntax:

```
for (variable of iterable) {  
  // code block to be executed  
}
```

variable - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with `const`, `let`, or `var`.

iterable - An object that has iterable properties.

Looping over an Array

Example

```
const cars = ["BMW", "Volvo", "Mini"];  
let text = "";  
  
for (let x of cars) {  
  text += x + " ";  
}
```

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

<h2>JavaScript For Of Loop</h2>
<p>The for of statement loops through the values of any iterable object:</p>

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

<script>
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x + "<br>";
}

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

</body>
</html>
```

JavaScript For Of Loop

The for of statement loops through the values of any iterable object:

BMW
Volvo
Mini

Looping over a String

Example

```
let language = "JavaScript";  
let text = "";  
  
for (let x of language) {  
    text += x + " ";  
}
```

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

<h2>JavaScript For Of Loop</h2>

<p>The for of statement loops through the values of an iterable object.</p>

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

<script>
let language = "JavaScript";

let text = "";
for (let x of language) {
  text += x + "<br>";
}

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

</body>
</html>
```

JavaScript For Of Loop

The for of statement loops through the values of an iterable object.

J
a
v
a
S
c
r
i
p
t

JavaScript Maps

Being able to use an Object as a key is an important Map feature.

Example

```
const fruits = new Map([  
  ["apples", 500],  
  ["bananas", 300],  
  ["oranges", 200]  
]);
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Map Objects</h2>
<p>Creating a Map from an Array:</p>

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

<script>
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

document.getElementById("demo").innerHTML = fruits.get("apples");
</script>

</body>
</html>
```

JavaScript Map Objects

Creating a Map from an Array:

500

JavaScript Classes

JavaScript Classes are templates for JavaScript Objects.

Use the keyword `class` to create a class.

Always add a method named `constructor()` :

Syntax

```
class ClassName {  
    constructor() { ... }  
}
```


The Symbol Type

A JavaScript Symbol is a primitive datatype just like Number, String, or Boolean.

It represents a unique "hidden" identifier that no other code can accidentally access.

For instance, if different coders want to add a `person.id` property to a person object belonging to a third-party code, they could mix each others values.

Using `Symbol()` to create a unique identifiers, solves this problem:

Example

```
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};  
  
let id = Symbol('id');  
person[id] = 140353;  
// Now person[id] = 140353  
// but person.id is still undefined
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Using JavaScript Symbol()</h2>
```

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

```
<script>
```

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

```
let id = Symbol('id');  
person[id] = 140353;
```

```
document.getElementById("demo").innerHTML = person[id] + " " + person.id;  
</script>
```

```
</body>
```

```
</html>
```

Using JavaScript Symbol()

140353 undefined

String.endsWith()

The `endsWith()` method returns `true` if a string ends with a specified value, otherwise `false` :

Example

```
var text = "John Doe";  
text.endsWith("Doe")    // Returns true
```

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

<h2>JavaScript Strings</h2>

<p>Check if a string ends with "Doe":</p>

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

<p>The endsWith() method is not supported in Internet Explorer.</p>

<script>
let text = "John Doe";
document.getElementById("demo").innerHTML = text.endsWith("Doe");
</script>

</body>
</html>
```

JavaScript Strings

Check if a string ends with "Doe":

true

The endsWith() method is not supported in Internet Explorer.

Array.from()

The `Array.from()` method returns an Array object from any object with a length property or any iterable object.

Example

Create an Array from a String:

```
Array.from("ABCDEFGH") // Returns [A,B,C,D,E,F,G,H]
```

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

<h2>JavaScript Arrays</h2>

<p>The Array.from() method returns an Array object from any object with a length property
or any iterable object.</p>

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

<script>
const myArr = Array.from("ABCDEFGH");
document.getElementById("demo").innerHTML = myArr;
</script>

<p>The Array.from() method is not supported in Internet Explorer.</p>

</body>
</html>
```

JavaScript Arrays

The Array.from() method returns an Array object from any object with a length property or any iterable object.

A,B,C,D,E,F,G

The Array.from() method is not supported in Internet Explorer.

Array keys()

The `keys()` method returns an Array Iterator object with the keys of an array.

Example

Create an Array Iterator object, containing the keys of the array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = fruits.keys();

let text = "";
for (let x of keys) {
  text += x + "<br>";
}
```

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

<h2>JavaScript Arrays</h2>

<p>The Array.keys() method returns an Array Iterator object with the keys of the array.
</p>

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

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

let text = "";
for (let x of keys) {
  text += x + "<br>";
}

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

<p>Array.keys() is not supported in Internet Explorer.</p>

</body>
</html>
```

JavaScript Arrays

The Array.keys() method returns an Array Iterator object with the keys of the array.

0
1
2
3

Array.keys() is not supported in Internet Explorer.

Array find()

The `find()` method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

Example

```
const numbers = [4, 9, 16, 25, 29];  
let first = numbers.find(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}
```

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

<h2>JavaScript Array.find()</h2>
<p id="demo"></p>

<script>
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

document.getElementById("demo").innerHTML = "First number over 18 is " + first;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.find()

First number over 18 is 25

Array findIndex()

The `findIndex()` method returns the index of the first array element that passes a test function.

This example finds the index of the first element that is larger than 18:

Example

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

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

<h2>JavaScript Array.findIndex()</h2>
<p id="demo"></p>

<script>
const numbers = [4, 9, 16, 25, 29];

document.getElementById("demo").innerHTML = "First number over 18 has index " +
numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.findIndex()

First number over 18 has index 3

New Math Methods

ES6 added the following methods to the Math object:

- `Math.trunc()`
- `Math.sign()`
- `Math.cbrt()`
- `Math.log2()`
- `Math.log10()`

The Math.trunc() Method

`Math.trunc(x)` returns the integer part of x:

Example

```
Math.trunc(4.9);    // returns 4
Math.trunc(4.7);    // returns 4
Math.trunc(4.4);    // returns 4
Math.trunc(4.2);    // returns 4
Math.trunc(-4.2);   // returns -4
```

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

<h2>JavaScript Math.trunc()</h2>

<p>Math.trunc(x) returns the integer part of x:</p>

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

<script>
document.getElementById("demo").innerHTML = Math.trunc(4.7);
</script>

</body>
</html>
```

JavaScript Math.trunc()

Math.trunc(x) returns the integer part of x:

4

The Math.sign() Method

`Math.sign(x)` returns if x is negative, null or positive:

Example

```
Math.sign(-4);    // returns -1  
Math.sign(0);     // returns 0  
Math.sign(4);     // returns 1
```

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

<h2>JavaScript Math.sign()</h2>

<p>Math.sign(x) returns if x is negative, null or positive:</p>

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

<script>
document.getElementById("demo").innerHTML = Math.sign(4);
</script>

</body>
</html>
```

JavaScript Math.sign()

Math.sign(x) returns if x is negative, null or positive:

1

The Math.log2() Method

`Math.log2(x)` returns the base 2 logarithm of x:

Example

```
Math.log2(2);    // returns 1
```

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

<h2>JavaScript Math.log2()</h2>

<p>Math.log2() returns the base 2 logarithm of a number.</p>
<p>How many times must we multiply 2 to get 8?</p>

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

<script>
document.getElementById("demo").innerHTML = Math.log2(8);
</script>

</body>
</html>
```

JavaScript Math.log2()

Math.log2() returns the base 2 logarithm of a number.

How many times must we multiply 2 to get 8?

3

New Number Properties

ES6 added the following properties to the Number object:

- `EPSILON`
- `MIN_SAFE_INTEGER`
- `MAX_SAFE_INTEGER`

New Number Methods

ES6 added 2 new methods to the Number object:

- `Number.isInteger()`
- `Number.isSafeInteger()`

The Number.isInteger() Method

The `Number.isInteger()` method returns `true` if the argument is an integer.

Example

```
Number.isInteger(10);    // returns true
Number.isInteger(10.5);  // returns false
```

The Number.isSafeInteger() Method

A safe integer is an integer that can be exactly represented as a double precision number.

The `Number.isSafeInteger()` method returns `true` if the argument is a safe integer.

Example

```
Number.isSafeInteger(10);    // returns true  
Number.isSafeInteger(12345678901234567890); // returns false
```

New Global Methods

ES6 added 2 new global number methods:

- `isFinite()`
- `isNaN()`

The `isFinite()` Method

The global `isFinite()` method returns `false` if the argument is `Infinity` or `NaN`.

Otherwise it returns `true`:

Example

```
isFinite(10/0);    // returns false
isFinite(10/1);    // returns true
```

Modules

Modules are imported in two different ways:

Import from named exports

Import named exports from the file person.js:

```
import { name, age } from './person.js';
```