w3schools.com
THE WORLD'S LARGEST WEB DEVELOPER SITE
HTML CSS JAVASCRIPT MORE REFERENCES

ECMAScript 6 - ECMAScript 2015

❮ Previous Next ❯

What is ECMAScript 6?

ECMAScript 6 is also known as ES6 and ECMAScript 2015.

Some people call it JavaScript 6.

This chapter will introduce some of the new features in ES6.

- JavaScript `let`
- JavaScript `const`
- Exponentiation (`**`) (EcmaScript 2016)
- Default parameter values
- `Array.find()`
- `Array.findIndex()`

Browser Support for ES6 (ECMAScript 2015)

Safari 10 and Edge 14 were the first browsers to fully support ES6:

				
Chrome 58	Edge 14	Firefox 54	Safari 10	Opera 55
Jan 2017	Aug 2016	Mar 2017	Jul 2016	Aug 2018

JavaScript let

The `let` statement 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
```

Try it Yourself »

JavaScript const

The `const` statement 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
```

Try it Yourself »

Read more about `let` and `const` in our JavaScript Let / Const Chapter.

Exponentiation Operator

The **exponentiation** operator (`**`) raises the first operand to the power of the second operand.

Example

```
var x = 5;
var z = x ** 2;          // result is 25
```

Try it Yourself »

`x ** y` produces the same result as `Math.pow(x,y)` :

Example

```
var x = 5;
var z = Math.pow(x,2);   // result is 25
```

Try it Yourself »

Default Parameter Values

ES6 allows function parameters to have default values.

Example

```
function myFunction(x, y = 10) {
  // y is 10 if not passed or undefined
  return x + y;
}
myFunction(5); // will return 15
```

Try it Yourself »

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

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

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

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index

- The array itself

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

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

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

Try it Yourself »

Note that the function takes 3 arguments:

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

New Number Properties

ES6 added the following properties to the Number object:

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

Example

```
var x = Number.EPSILON;
```

Try it Yourself »

Example

```
var x = Number.MIN_SAFE_INTEGER;
```

Try it Yourself »

Example

```
var x = Number.MAX_SAFE_INTEGER;
```

Try it Yourself »

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
```

Try it Yourself »

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
```

Try it Yourself »

Safe integers are all integers from $-(2^{53} - 1)$ to $+(2^{53} - 1)$.
This is safe: 9007199254740991. This is not safe: 9007199254740992.

New Global Methods

ES6 also 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
```

Try it Yourself »

The isNaN() Method

The global `isNaN()` method returns `true` if the argument is `NaN`. Otherwise it returns `false`:

Example

```
isNaN("Hello");       // returns true
```

Try it Yourself »

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;
```

Try it Yourself »

Arrow functions do not have their own `this`. They are not well suited for defining **object methods**.

Arrow functions are not hoisted. They must be defined **before** they are used.

Using `const` is safer than using `var`, because a function expression is always constant value.

You can only omit the `return` keyword and the curly brackets if the function is a single statement. Because of this, it might be a good habit to always keep them:

Example

```
const x = (x, y) => { return x * y };
```

Try it Yourself »

❮ Previous

Next ❯