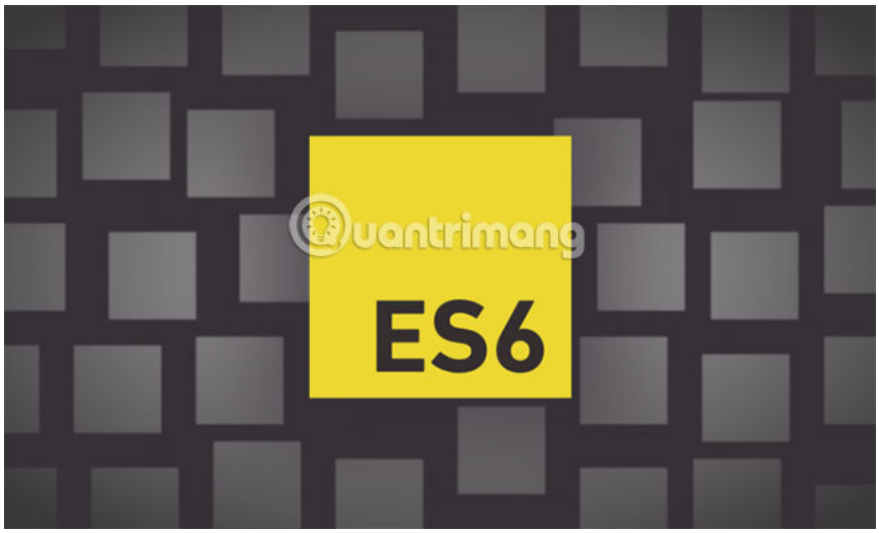
Learn about ES6 in Javascript

ES6 refers to version 6 of the ECMA Script programming language. ECMA Script is the standard name for JavaScript and version 6 is the next version after version 5, released in 2011.

ES6 refers to version 6 of the ECMA Script programming language. ECMA Script is the standard name for JavaScript and version 6 is the next version after version 5 released in 2011. This is a major improvement for the JavaScript language and more features to help software development process Large tissue easier.

ECMAScript, or ES6, was published in June 2015. It was later renamed to ECMAScript 2015. Full web browser support for the language is still incomplete, although support for the main sections was conducted. The main web browsers support a number of ES6 features. However, software called converters can be used to convert ES6 code into ES5, because ES5 is better supported on most browsers.



Now, let's look at some of the big changes that ES6 brings to JavaScript.

Learn about ES6 in Javascript

1. 1. Constant
2. 2. Variables and functions have block scope
3. 3. Arrow Function (Arrow function)
4. 4. The default function parameters
5. 5. The remaining function parameters
6. 6. String templating

7. 7. Object properties
8. 8. Syntax of defining the official class
    1. Define the class
    2. Method of declaration
    3. Getter and Setter
    4. Inheritance

1. Constant

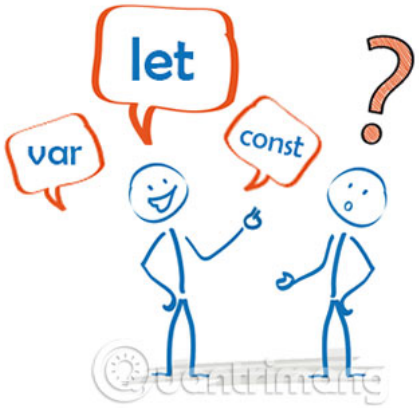
Finally the concept of constants has been included in JavaScript! Constants are values ??that can only be specified once (in each range explained below). Two values ??for a constant in the same range will produce an error.

```
const JOE = 4.0
JOE = 3.5
// results in: Uncaught TypeError: Assignment to constant variable.
```

You can use constants anywhere you use a variable **(var).**

```
console.log ("Value is:" + joe * 2)
// prints: 8
```

2. Variables and functions have block scope



With ES6, declared variables use **let** (and the constants described above), follow the block range rules like in Java, C ++, etc.

Before this update, variables in JavaScript are **scoped** functions **.** That is, when you need a new scope for a variable, you must declare it in a function.

Variables keep the value until the end of the block. Then, the value in the external block (if any) is restored.

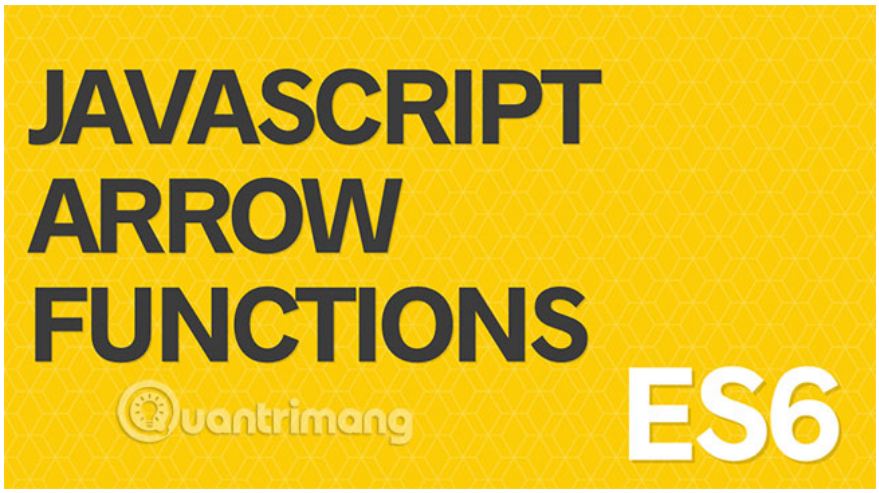
```
{
let x = "hello";
{
let x = "world";
```

```
console.log ("inner block, x =" + x);
}
console.log ("outer block, x =" + x);
}
// prints
inner block, x = world
outer block, x = hello
```

You can also redefine constants in such blocks.

```
{
let x = "hello";
{
const x = 4.0;
console.log ("inner block, x =" + x);
try {
x = 3.5
} catch (err) {
console.error ("inner block:" + err);
}
}
x = "world";
console.log ("outer block, x =" + x);
}
// prints
inner block, x = 4
inner block: TypeError: Assignment ?? kích c? constant.
outer block, x = world
```

3. Arrow Function (Arrow function)



ES6 offers a new syntax for defining functions using an arrow. In the following example, **x** is a function that accepts a parameter called **a** and returns its value:

```
var x = a => a + 1;
x (4) // returns 5
```

Using this syntax, you can easily define and pass arguments in functions. Please use **forEach ():**

```
[1, 2, 3, 4] .forEach (a => console.log (a + "=>" + a * a))
// prints
1 => 1
2 => 4
3 => 9
4 => 16
```

Define functions that accept multiple arguments by placing them in parentheses:

```
[22, 98, 3, 44, 67] .sort ((a, b) => a - b)
// returns
[3, 22, 44, 67, 98]
```

4. The default function parameters

Function parameters can be declared with default values. In the following example, **x** is a function that has two parameters **a** and **b** . The second parameter - **b** is assigned a default value of **1** .

```
var x = (a, b = 1) => a * b
x (2)
// returns 2
x (2, 2)
// returns 4
```

Unlike other languages ??such as C ++ or Python, parameters with default values ??may appear before parameters without default values. Note that this function is defined as a block with **return** value as follows:

```
var x = (a = 2, b) => {return a * b}
```

However, the argument is combined from left to right. After being first called as the example below, **b** has an unknown value, although it has been declared with a default value. The passed argument will match **a** instead of **b** . The function returns **NaN.**

```
x (2)
// returns NaN
x (1, 3)
// returns 3
```

When you convert **undefined** as an argument, the default value is used if available.

```
x (undefined, 3)
// returns 6
```

5. The remaining function parameters

When calling a function, sometimes there is a need to pass, to pass an arbitrary number of arguments and handle these arguments in the function. This need is handled by the remaining function parameters. This is the way to capture the rest of the arguments, followed by other arguments defined using the syntax shown below. These arguments are recorded in an array.

```
var x = function (a, b, . args) {console.log ("a =" + a + ", b =" + b + "," + a
x (2, 3)
// prints
a = 2, b = 3, 0 args left
x (2, 3, 4, 5)
// prints
a = 2, b = 3, 2 args left
```

6. String templating

String templating refers to interpolation of variables and expressions into strings, using a syntax like **perl** or **shell.** A string template is enclosed in backslash characters (`). In contrast, single quotes (') or quotation marks (') indicate normal strings. The inner expression of the form is marked from **$ {** and **}** . Here is an example:

```
var name = "joe";
var x = `hello $ {name}`
// returns "hello joe"
```

Of course, you can use an arbitrary expression to evaluate.

```
// define an arrow function
var f = a => a * 4

// ??t m?t giá tr? tham s?
var v = 5

// và x? lý các function trong m?u chu?i
var x = `hello $ {f (v)}`
// returns "hello 20"
```

The syntax for defining this string can also be used to identify multiple lines.

```
var x = `hello world
next line`
// returns
hello world
next line
```

7. Object properties

ES6 provides a simple object creation syntax. See the example below:

```
var x = "hello world", y = 25
var a = {x, y}
// is equivalent to the ES5:
{x: x, y: y}
```

Attribute names **computed** quite conveniently. For ES5 and earlier versions, to set the object attribute to a computed name, you must do this:

```
var x = "hello world", y = 25
var a = {x: x, y: y}
a ["joe" + y] = 4
// a is now:
{x: "hello world", y: 25, joe25: 4}
```

Now you can do it all in a single determination step:

```
var a = {x, y, ["joe" + y]: 4}
// returns
{x: "hello world", y: 25, joe25: 4}
```

And of course, to define methods, you can only specify it by name:

```
var a = {x, y, ["joe" + y]: 4, foo (v) {return v + 4}}
a.foo (2)
// returns
6
```

8. Syntax of defining the official class

Define the class

And finally, JavaScript provides a formal class definition syntax. While it is merely a syntax created from classes based on the existing prototype, it serves to enhance code clarity. That means it doesn't add a new object or anything like that.

```
class Circle {
constructor (radius) {
this.radius = radius
}
}
// use it
var c = new Circle (4)
// returns: Circle {radius: 4}
```

Method of declaration

Determining a method is also quite simple.

```
class Circle {
constructor (radius) {
this.radius = radius
}
computeArea () {return Math.PI * this.radius * this.radius}
}
var c = new Circle (4)
c.computeArea ()
// returns: 50.26548245743669
```

Getter and Setter

Now, we have **getter** and **setter,** with a simple update to the syntax. **Redefine** the **Circle** class with the **area** attribute **.**

```
class Circle {
constructor (radius) {
this.radius = radius
}
get area () {return Math.PI * this.radius * this.radius}
}
var c = new Circle (4)
// returns: Circle {radius: 4}
c.area
// returns: 50.26548245743669
```

Now, add a setter. In order to be able to define a **radius** as a preset attribute, we will redefine the actual field to **_radius** or something that will not conflict with the setter. If not, we will see ' **stack overflow error** '.

This is the redefined class:

```
class Circle {
constructor (radius) {
this._radius = radius
}
get area () {return Math.PI * this._radius * this._radius}
radius (r) set {this._radius = r}
}
var c = new Circle (4)
// returns: Circle {_radius: 4}
c.area
// returns: 50.26548245743669
c.radius = 6
c.area
// returns: 113.09733552923255
```

Overall, this is a pretty good addition to object-oriented JavaScript.

Inheritance

In addition to defining classes using the **class** keyword **,** you can also use the **extends** keyword to inherit from super classes. Let's see how to do this with an example.

```
class Ellipse {
constructor (width, height) {
this._width = width;
this._height = height;
}
get area () {return Math.PI * this._width * this._height; } }
set width (w) {this._width = w; } }
set height (h) {this._height = h; } }
}

class Circle extends Ellipse {
constructor (radius) {
```

```
super (radius, radius);
}
set radius (r) {super.width = r; super.height = r; super.height = r; } }
}

// create a circle
var c = new Circle (4)
// returns: Circle {_width: 4, _height: 4}
c.radius = 2
// c is now: Circle {_width: 2, _height: 2}
c.area
// returns: 12.566370614359172
c.radius = 5
c.area
// returns: 78.53981633974483
```

Above is a brief introduction to some of the features of JavaScript ES6.

Are you using ES6 in your projects? How is your experience? Please let us know in the comment section below!

See more:

1. 12 extremely useful tricks for JavaScript programmers
2. Udemy's top 5 JavaScript courses
3. 6 best free tutorials to learn about React and create web applications