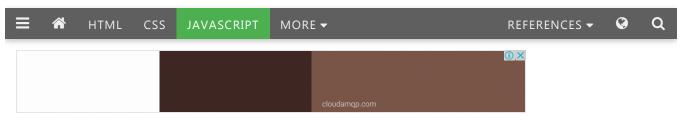
w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE



ECMAScript 5 - JavaScript 5



What is ECMAScript 5?

ECMAScript 5 is also known as ES5 and ECMAScript 2009

This chapter introduces some of the most important features of ES5.

ECMAScript 5 Features

These were the new features released in 2009:

- The "use strict" Directive
- String.trim()
- Array.isArray()
- Array.forEach()
- Array.map()
- Array.filter()
- Array.reduce()
- Array.reduceRight()
- Array.every()
- Array.some()
- Array.indexOf()
- Array.lastIndexOf()
- JSON.parse()
- JSON.stringify()
- Date.now()
- Property Getters and Setters
- New Object Property Methods

ECMAScript 5 Syntactical Changes

- Property access [] on strings
- Trailing commas in array and object literals

- Multiline string literals
- Reserved words as property names

The "use strict" Directive

"use strict" defines that the JavaScript code should be executed in "strict mode".

With strict mode you can, for example, not use undeclared variables.

You can use strict mode in all your programs. It helps you to write cleaner code, like preventing you from using undeclared variables.

"use strict" is just a string expression. Old browsers will not throw an error if they don't understand it.

Read more in JS Strict Mode.

String.trim()

String.trim() removes whitespace from both sides of a string.

```
Example

var str = " Hello World! ";
alert(str.trim());

Try it Yourself >>
```

Read more in JS String Methods.

Array.isArray()

The isArray() method checks whether an object is an array.

```
function myFunction() {
   var fruits = ["Banana", "Orange", "Apple", "Mango"];
   var x = document.getElementById("demo");
   x.innerHTML = Array.isArray(fruits);
}
Try it Yourself >>
```

Read more in JS Arrays.

Array.forEach()

The forEach() method calls a function once for each array element.

```
Example

var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value) {
   txt = txt + value + "<br>;
}
Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.map()

This example multiplies each array value by 2:

```
Example

var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
Try it Yourself »
```

Learn more in <u>JS Array Iteration Methods</u>.



Array.filter()

This example creates a new array from elements with a value larger than 18:

```
Example

var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value) {
   return value > 18;
}
Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.reduce()

This example finds the sum of all numbers in an array:

```
Example

var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value) {
   return total + value;
}
Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.reduceRight()

This example also finds the sum of all numbers in an array:

```
Example

var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduceRight(myFunction);

function myFunction(total, value) {
   return total + value;
}
Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.every()

This example checks if all values are over 18:

```
Example

var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value) {
   return value > 18;
}
Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.some()

This example checks if some values are over 18:

```
Example

var numbers = [45, 4, 9, 16, 25];
var all0ver18 = numbers.some(myFunction);

function myFunction(value) {
   return value > 18;
}
Try it Yourself »
```

Learn more in <u>JS Array Iteration Methods</u>.

Array.indexOf()

Search an array for an element value and returns its position.

```
Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");

Try it Yourself »
```

Learn more in JS Array Iteration Methods.

Array.lastIndexOf()

Array.lastIndexOf() is the same as Array.indexOf(), but searches from the end of the array.

```
Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf("Apple");

Try it Yourself »
```

Learn more in JS Array Iteration Methods.

JSON.parse()

A common use of JSON is to receive data from a web server.

Imagine you received this text string from a web server:

```
'{"name":"John", "age":30, "city":"New York"}'
```

The JavaScript function JSON.parse() is used to convert the text into a JavaScript object:

```
var obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}');
Try it Yourself >>
```

Read more in our JSON Tutorial.

JSON.stringify()

A common use of JSON is to send data to a web server.

When sending data to a web server, the data has to be a string.

Imagine we have this object in JavaScript:

```
var obj = {"name":"John", "age":30, "city":"New York"};
```

Use the JavaScript function JSON.stringify() to convert it into a string.

```
var myJSON = JSON.stringify(obj);
```

The result will be a string following the JSON notation.

myJSON is now a string, and ready to be sent to a server:

```
var obj = {"name":"John", "age":30, "city":"New York"};
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
Try it Yourself »
```

Read more in our JSON Tutorial.

Date.now()

Date.now() returns the number of milliseconds since zero date (January 1. 1970 00:00:00 UTC).

```
Example
var timInMSs = Date.now();

Try it Yourself >>
```

Date.now() returns the same as getTime() performed on a Date object.

Learn more in JS Dates.

Property Getters and Setters

ES5 lets you define object methods with a syntax that looks like getting or setting a property.

This example creates a **getter** for a property called fullName:

```
Example

// Create an object:
var person = {
    firstName: "John",
    lastName : "Doe",
    get fullName() {
        return this.firstName + " " + this.lastName;
    }
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.fullName;
Try it Yourself »
```

This example creates a **setter** and a **getter** for the language property:

```
Example
 var person = {
   firstName: "John",
   lastName : "Doe",
   language : "NO",
   get lang() {
    return this.language;
   },
   set lang(value) {
     this.language = value;
   }
 };
 // Set an object property using a setter:
 person.lang = "en";
 // Display data from the object using a getter:
 document.getElementById("demo").innerHTML = person.lang;
 Try it Yourself »
```

This example uses a setter to secure upper case updates of language:

```
var person = {
  firstName: "John",
  lastName : "Doe",
```

```
language : "NO",
set lang(value) {
   this.language = value.toUpperCase();
}
};

// Set an object property using a setter:
person.lang = "en";

// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
Try it Yourself »
```

Learn more about Gettes and Setters in JS Object Accessors

New Object Property Methods

Object.defineProperty() is a new Object method in ES5.

It lets you define an object property and/or change a property's value and/or metadata.

```
Example
 // Create an Object:
 var person = {
   firstName: "John",
   lastName : "Doe",
   language : "NO",
 // Change a Property:
 Object.defineProperty(person, "language", {
   value: "EN",
   writable : true,
   enumerable : true,
   configurable : true
 // Enumerate Properties
 var txt = "";
 for (var x in person) {
   txt += person[x] + "<br>";
 document.getElementById("demo").innerHTML = txt;
 Try it Yourself »
```

Next example is the same code, except it hides the language property from enumeration:

```
Example
 // Create an Object:
 var person = {
   firstName: "John",
   lastName : "Doe",
   language : "NO",
 // Change a Property:
 Object.defineProperty(person, "language", {
   value: "EN",
   writable : true,
   enumerable : false,
   configurable : true
 });
 // Enumerate Properties
 var txt = "";
 for (var x in person) {
   txt += person[x] + "<br>";
 document.getElementById("demo").innerHTML = txt;
```

This example creates a setter and a getter to secure upper case updates of language:

```
Example

/// Create an Object:
var person = {
    firstName: "John",
    lastName : "Doe",
    language : "NO"
};

// Change a Property:
Object.defineProperty(person, "language", {
    get : function() { return language },
    set : function(value) { language = value.toUpperCase()}
});

// Change Language
person.language = "en";

// Display Language
```

```
document.getElementById("demo").innerHTML = person.language;

Try it Yourself »
```

ECMAScript 5 added a lot of new Object Methods to JavaScript:

```
ES5 New Object Methods
 // Adding or changing an object property
 Object.defineProperty(object, property, descriptor)
 // Adding or changing many object properties
 Object.defineProperties(object, descriptors)
 // Accessing Properties
 Object.getOwnPropertyDescriptor(object, property)
 // Returns all properties as an array
 Object.getOwnPropertyNames(object)
 // Returns enumerable properties as an array
 Object.keys(object)
 // Accessing the prototype
 Object.getPrototypeOf(object)
 // Prevents adding properties to an object
 Object.preventExtensions(object)
 // Returns true if properties can be added to an object
 Object.isExtensible(object)
 // Prevents changes of object properties (not values)
 Object.seal(object)
 // Returns true if object is sealed
 Object.isSealed(object)
 // Prevents any changes to an object
 Object.freeze(object)
 // Returns true if object is frozen
 Object.isFrozen(object)
```

Learn more in Object ECMAScript5.

Property Access on Strings

The charAt() method returns the character at a specified index (position) in a string:

```
Example

var str = "HELLO WORLD";
str.charAt(0);  // returns H

Try it Yourself »
```

ECMAScript 5 allows property access on strings:

Property access on string might be a little unpredictable.

Read more in JS String Methods.

Trailing Commas

ECMAScript 5 allows trailing commas in object and array definitions:

```
Object Example

person = {
  firstName: "John",
  lastName: " Doe",
  age: 46,
}
```

```
Array Example

points = [
    1,
    5,
    10,
    25,
    40,
    100,
    ];
```

```
WARNING !!!

Internet Explorer 8 will crash.

JSON does not allow trailing commas.
```

```
JSON Objects:

// Allowed:
var person = '{"firstName":"John", "lastName":"Doe", "age":46}'
JSON.parse(person)

// Not allowed:
var person = '{"firstName":"John", "lastName":"Doe", "age":46,}'
JSON.parse(person)

JSON Arrays:

// Allowed:
points = [40, 100, 1, 5, 25, 10]

// Not allowed:
points = [40, 100, 1, 5, 25, 10,]
```

Strings Over Multiple Lines

ECMAScript 5 allows string literals over multiple lines if escaped with a backslash:

```
Example

"Hello \
Dolly!";
Try it Yourself >>
```

The $\$ method might not have universal support. Older browsers might treat the spaces around the backslash differently. Some older browsers do not allow spaces behind the $\$ character.

A safer way to break up a string literal, is to use string addition:

```
Example
"Hello " +
"Dolly!";
```

```
Try it Yourself »
```

Reserved Words as Property Names

ECMAScript 5 allows reserved words as property names:

```
Object Example

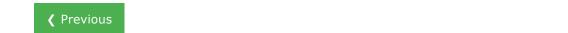
var obj = {name: "John", new: "yes"}

Try it Yourself »
```

Browser Support for ES5 (ECMAScript 5)

Chrome 23, IE 10, and Safari 6 were the first browsers to fully support ECMAScript 5:

O	9	⑤		0
Chrome 23	IE10 / Edge	Firefox 21	Safari 6	Opera 15
Sep 2012	Sep 2012	Apr 2013	Jul 2012	Jul 2013



Next >