Loops and Iteration

[**for statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#for_statement)

A [for](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for) loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop.

A for statement looks as follows:

JS

for (initialization; condition; afterthought)

statement

or

for (initialization; condition; afterthought){

statement

}

When a for loop executes, the following occurs:

1. The initializing expression initialization, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity. This expression can also declare variables.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. Otherwise, the for loop terminates. (If the condition expression is omitted entirely, the condition is assumed to be true.)
3. The statement executes. To execute multiple statements, use a [block statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block) ({ }) to group those statements.
4. If present, the update expression afterthought is executed.
5. Control returns to Step 2.

[**Example**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example)

In the example below, the function contains a for statement that counts the number of selected options in a scrolling list (a [<select>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select) element that allows multiple selections).

**HTML**

HTML

<form name="selectForm">

<label for="musicTypes"

>Choose some music types, then click the button below:</label

>

<select id="musicTypes" name="musicTypes" multiple>

<option selected>R&B</option>

<option>Jazz</option>

<option>Blues</option>

<option>New Age</option>

<option>Classical</option>

<option>Opera</option>

</select>

<button id="btn" type="button">How many are selected?</button>

</form>

**JavaScript**

Here, the for statement declares the variable i and initializes it to 0. It checks that i is less than the number of options in the <select> element, performs the succeeding if statement, and increments i by 1 after each pass through the loop.

JS

function countSelected(selectObject) {

let numberSelected = 0;

for (let i = 0; i < selectObject.options.length; i++) {

if (selectObject.options[i].selected) {

numberSelected++;

}

}

return numberSelected;

}

const btn = document.getElementById("btn");

btn.addEventListener("click", () => {

const musicTypes = document.selectForm.musicTypes;

console.log(`You have selected ${countSelected(musicTypes)} option(s).`);

});

[**do...while statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#do...while_statement)

The [do...while](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/do...while) statement repeats until a specified condition evaluates to false.

A do...while statement looks as follows:

JS

do

statement

while (condition):

or

do{

statement

}while (condition):

statement is always executed once before the condition is checked. (To execute multiple statements, use a block statement ({ }) to group those statements.)

If condition is true, the statement executes again. At the end of every execution, the condition is checked. When the condition is false, execution stops, and control passes to the statement following do...while.

[**Example**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_2)

In the following example, the do loop iterates at least once and reiterates until i is no longer less than 5.

JS

let i = 0;

do {

i += 1;

console.log(i);

} while (i < 5);

[**while statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#while_statement)

A [while](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/while) statement executes its statements as long as a specified condition evaluates to true. A while statement looks as follows:

JSCopy to Clipboard

while (condition)

statement

If the condition becomes false, statement within the loop stops executing and control passes to the statement following the loop.

The condition test occurs *before* statement in the loop is executed. If the condition returns true, statement is executed and the condition is tested again. If the condition returns false, execution stops, and control is passed to the statement following while.

To execute multiple statements, use a block statement ({ }) to group those statements.

[**Example 1**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_1)

The following while loop iterates as long as n is less than 3:

JS

let n = 0;

let x = 0;

while (n < 3) {

n++;

x += n;

}

With each iteration, the loop increments n and adds that value to x. Therefore, x and n take on the following values:

* After the first pass: n = 1 and x = 1
* After the second pass: n = 2 and x = 3
* After the third pass: n = 3 and x = 6

After completing the third pass, the condition n < 3 is no longer true, so the loop terminates.

[**Example 2**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_2_2)

Avoid infinite loops. Make sure the condition in a loop eventually becomes false—otherwise, the loop will never terminate! The statements in the following while loop execute forever because the condition never becomes false:

Js

// Infinite loops are bad!

while (true) {

console.log("Hello, world!");

}

[**labeled statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#labeled_statement)

A [label](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label) provides a statement with an identifier that lets you refer to it elsewhere in your program. For example, you can use a label to identify a loop, and then use the break or continue statements to indicate whether a program should interrupt the loop or continue its execution.

The syntax of the labeled statement looks like the following:

JS

label:

statement

or

The value of label may be any JavaScript identifier that is not a reserved word. The statement that you identify with a label may be any statement.

[**Example**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_3)

In this example, the label markLoop identifies a while loop.

JS

markLoop: while (theMark) {

doSomething();

}

[**break statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#break_statement)

Use the [break](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/break) statement to terminate a loop, switch, or in conjunction with a labeled statement.

* When you use break without a label, it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.
* When you use break with a label, it terminates the specified labeled statement.

The syntax of the break statement looks like this:

JS

break;

break label;

1. The first form of the syntax terminates the innermost enclosing loop or switch.
2. The second form of the syntax terminates the specified enclosing labeled statement.

[**Example 1**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_1_2)

The following example iterates through the elements in an array until it finds the index of an element whose value is theValue:

JS

for (let i = 0; i < a.length; i++) {

if (a[i] === theValue) {

break;

}

}

[**Example 2: Breaking to a label**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_2_breaking_to_a_label)

JS

let x = 0;

let z = 0;

labelCancelLoops: while (true) {

console.log("Outer loops:", x);

x += 1;

z = 1;

while (true) {

console.log("Inner loops:", z);

z += 1;

if (z === 10 && x === 10) {

break labelCancelLoops;

} else if (z === 10) {

break;

}

}

}

[**continue statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#continue_statement)

The [continue](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue) statement can be used to restart a while, do-while, for, or label statement.

* When you use continue without a label, it terminates the current iteration of the innermost enclosing while, do-while, or for statement and continues execution of the loop with the next iteration. In contrast to the break statement, continue does not terminate the execution of the loop entirely. In a while loop, it jumps back to the condition. In a for loop, it jumps to the increment-expression.
* When you use continue with a label, it applies to the looping statement identified with that label.

The syntax of the continue statement looks like the following:

JS

continue;

continue label;

[**Example 1**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_1_3)

The following example shows a while loop with a continue statement that executes when the value of i is 3. Thus, n takes on the values 1, 3, 7, and 12.

JS

let i = 0;

let n = 0;

while (i < 5) {

i++;

if (i === 3) {

continue;

}

n += i;

console.log(n);

}

//1,3,7,12

If you comment out the continue;, the loop would run till the end and you would see 1,3,6,10,15.

[**Example 2**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_2_3)

A statement labeled checkiandj contains a statement labeled checkj. If continue is encountered, the program terminates the current iteration of checkj and begins the next iteration. Each time continue is encountered, checkj reiterates until its condition returns false. When false is returned, the remainder of the checkiandj statement is completed, and checkiandj reiterates until its condition returns false. When false is returned, the program continues at the statement following checkiandj.

If continue had a label of check I and j, the program would continue at the top of the check I and j statement.

JS

let i = 0;

let j = 10;

checkiandj: while (i < 4) {

console.log(i);

i += 1;

checkj: while (j > 4) {

console.log(j);

j -= 1;

if (j % 2 === 0) {

continue checkj;

}

console.log(j, "is odd.");

}

console.log("i =", i);

console.log("j =", j);

}

[**for...in statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#for...in_statement)

The [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) statement iterates a specified variable over all the enumerable properties of an object. For each distinct property, JavaScript executes the specified statements. A for...in statement looks as follows:

JS

for (variable in object)

statement

[**Example**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#example_4)

The following function takes as its argument an object and the object's name. It then iterates over all the object's properties and returns a string that lists the property names and their values.

JS

function dumpProps(obj, objName) {

let result = "";

for (const i in obj) {

result += `${objName}.${i} = ${obj[i]}<br>`;

}

result += "<hr>";

return result;

}

For an object car with properties make and model, result would be:

car.make = Ford

car.model = Mustang

[**Arrays**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#arrays)

Although it may be tempting to use this as a way to iterate over [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) elements, the for...in statement will return the name of your user-defined properties in addition to the numeric indexes.

Therefore, it is better to use a traditional [for](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for) loop with a numeric index when iterating over arrays, because the for...in statement iterates over user-defined properties in addition to the array elements, if you modify the Array object (such as adding custom properties or methods).

[**for...of statement**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#for...of_statement)

The [for...of](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of) statement creates a loop Iterating over [iterable objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols) (including [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array), [Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map), [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set), [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) object and so on), invoking a custom iteration hook with statements to be executed for the value of each distinct property.

JS

for (variable of object)

statement

The following example shows the difference between a for...of loop and a [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loop. While for...in iterates over property names, for...of iterates over property values:

JS

const arr = [3, 5, 7];

arr.foo = "hello";

for (const i in arr) {

console.log(i);

}

// "0" "1" "2" "foo"

for (const i of arr) {

console.log(i);

}

// Logs: 3 5 7

The for...of and for...in statements can also be used with [destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment). For example, you can simultaneously loop over the keys and values of an object using [Object.entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/entries).

JS

const obj = { foo: 1, bar: 2 };

for (const [key, val] of Object.entries(obj)) {

console.log(key, val);

}

// "foo" 1

// "bar" 2