**Loops: while and for**

We often have a need to perform similar actions many times in a row.

For example, when we need to output goods from the list one after another. Or just run the same code for each number from 1 to 10.

*Loops* are a way to repeat the same part of code multiple times.

**[The “while” loop](https://javascript.info/while-for" \l "the-while-loop)**

The while loop has the following syntax:

while (condition) {

// code

// so-called "loop body"

}

While the condition is true – the code from the loop body is executed.

For instance, the loop below outputs i while i<3:

let i = 0;

while (i < 3) { // shows 0, then 1, then 2

alert( i );

i++;

}

A single execution of the loop body is called *an iteration*. The loop in the example above makes 3 iterations.

If there were no i++ in the example above, the loop would repeat (in theory) forever. In practice, the browser provides ways to stop such loops, and for server-side JavaScript we can kill the process.

**[The “do…while” loop](https://javascript.info/while-for" \l "the-do-while-loop)**

The condition check can be moved *below* the loop body using the do..while syntax:

do {

// loop body

} while (condition);

The loop will first execute the body, then check the condition, and while it’s truthy – execute it again and again.

For example:

let i = 0;

do {

alert( i );

i++;

} while (i < 3);

This form of syntax is rarely used except when you want the body of the loop to execute **at least once** regardless of the condition being truthy. Usually, the other form is preferred: while(…) {…}.

[**The “for” loop**](https://javascript.info/while-for#the-for-loop)

The for loop is the most often used one.

It looks like this:

for (begin; condition; step) {

// ... loop body ...

}

Let’s learn the meaning of these parts by example. The loop below runs alert(i) for i from 0 up to (but not including) 3:

for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2

alert(i);

}

Let’s examine the for statement part by part:

| **part** |  |  |
| --- | --- | --- |
| begin | i=0 | Executes once upon entering the loop. |
| condition | i<3 | Checked before every loop iteration, if fails the loop stops. |
| step | i++ | Executes after the body on each iteration, but before the condition check. |
| body | alert(i) | Runs again and again while the condition is truthy |

The general loop algorithm works like this:

Run begin

→ (if condition → run body and run step)

→ (if condition → run body and run step)

→ (if condition → run body and run step)

→ ...

If you are new to loops, then maybe it would help if you go back to the example and reproduce how it runs step-by-step on a piece of paper.

Here’s what exactly happens in our case:

// for (let i = 0; i < 3; i++) {alert(i);}

// run begin

let i = 0

// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// ...finish, because now i == 3

**Inline variable declaration**

Here the “counter” variable i is declared right in the loop. That’s called an “inline” variable declaration. Such variable is visible only inside the loop.

for (let i = 0; i < 3; i++) {

alert(i); // 0, 1, 2

}

alert(i); // error, no such variable

Instead of defining a variable, we can use an existing one:

let i = 0;

for (i = 0; i < 3; i++) { // use an existing variable

alert(i); // 0, 1, 2

}

alert(i); // 3, visible, because declared outside of the loop

**[Skipping parts](https://javascript.info/while-for" \l "skipping-parts)**

Any part of for can be skipped.

For example, we can omit begin if we don’t need to do anything at the loop start.

Like here:

let i = 0; // we have i already declared and assigned

for (; i < 3; i++) { // no need for "begin"

alert( i ); // 0, 1, 2

}

We can also remove the step part:

let i = 0;

for (; i < 3;) {

alert( i );

}

The loop became identical to while (i<3).

We can actually remove everything, thus creating an infinite loop:

for (;;) {

// repeats without limits

}

Please note that the two for semicolons ; must present, otherwise it would be a syntax error.

**[Breaking the loop](https://javascript.info/while-for" \l "breaking-the-loop)**

Normally the loop exits when the condition becomes falsy.

But we can force the exit at any moment. There’s a special break directive for that.

For example, this code below asks user for numbers and breaks if no number entered:

let sum = 0;

while (true) {

let value = +prompt("Enter a number", '');

if (!value) break; // (\*)

sum += value;

}

alert( 'Sum: ' + sum );

The break directive is activated in the line (\*) if the user enters an empty line or cancels the input. It stops the loop immediately, passing the control to the first line after the loop. Namely, alert.

The combination: “infinite loop + break as needed” is great for situations when the condition must be checked not in beginning/end of the loop, but in the middle. Or even in several places of the body.

**[Continue to the next iteration](https://javascript.info/while-for" \l "continue)**

The continue directive is a “lighter version” of break. It doesn’t stop the whole loop. Instead it stops the current iteration and forces the loop to start a new one (if the condition allows).

We can use it if we’re done on the current iteration and would like to move on to the next.

The loop above uses continue to output only odd values:

for (let i = 0; i < 10; i++) {

// if true, skip the remaining part of the body

if (i % 2 == 0) continue;

alert(i); // 1, then 3, 5, 7, 9

}

For even values of i the continue directive stops body execution, passing the control to the next iteration of for (with the next number). So the alert is only called for odd values.

**The directive continue helps to decrease nesting level**

A loop that shows odd values could look like this:

for (let i = 0; i < 10; i++) {

if (i % 2) {

alert( i );

}

}

From the technical point of view it’s identical to the example above. Surely, we can just wrap the code in the if block instead of continue.

But as a side-effect we got one more figure brackets nesting level. If the code inside if is longer than a few lines, that may decrease the overall readability.

**[Labels for break/continue](https://javascript.info/while-for" \l "labels-for-break-continue)**

Sometimes we need to break out from multiple nested loops at once.

For example, in the code below we loop over i and j prompting for coordinates (i, j) from (0,0) to (3,3):

for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

let input = prompt(`Value at coords (${i},${j})`, '');

// what if I want to exit from here to Done (below)?

}

}

alert('Done!');

We need a way to stop the process if the user cancels the input.

The ordinary break after input would only break the inner loop. That’s not sufficient. Labels come to the rescue.

A *label* is an identifier with a colon before a loop:

labelName: for(...) {

...

}

The break <labelName> statement in the loop breaks out to the label.

Like here:

outer: for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

let input = prompt(`Value at coords (${i},${j})`, '');

// if an empty string or canceled, then break out of both loops

if (!input) break outer; // (\*)

// do something with the value...

}

}

alert('Done!');

In the code above break outer looks upwards for the label named outer and breaks out of that loop.

So the control goes straight from (\*) to alert('Done!').

We can also move a label into the separate string:

outer:

for (let i = 0; i < 3; i++) { ... }

The continue directive can also be used with a label. In this case the execution jumps to the next iteration of the labelled loop.

[**Summary**](https://javascript.info/while-for#summary)

We covered 3 types of loops:

* while – the condition is checked before each iteration.
* do..while – the condition is checked after each iteration.
* for(;;) – the condition is checked before each iteration, additional settings available.

To make an “infinite” loop, usually the while(true) construct is used. Such a loop, just like any other, can be stopped with the break directive.

If we don’t want to do anything on the current iteration and would like to forward to the next one – the continue directive does it.

Break/continue support labels before the loop. A label is the only way for break/continue to escape the nesting and go to the outer loop.

**Questions and Exercises (loops.js)**

1. Use the for loop to output even numbers from 2 to 10.
2. Write a guessing game where the player must guess a computer generated number between 1 and 100. The user will have up to 10 guesses. After each guess provide the user with a hint as to how close they are to the actual number i.e. if they are 5 or less away from the actual number tell them their guess was hot, 6-10 away is warm and anything else is cold. Use a loop to help solve this problem.
3. The population of insects in a particular forest is estimated to be 12 960 000 000. A fungi has infected the forest and is now reducing this population at an average rate of 4.5% each year. Write a program that will determine how many years until all insects are wiped out.
4. Using a while loop write a program that asks a user for a positive number repeatedly until they enter a negative number. Display how many positive numbers were entered at the end.
5. Modify the last program so that you find the sum of all the positive numbers entered.
6. Further modify this program so that the average of all positive numbers entered is displayed.
7. Finally. Modify the program one last time so that the largest and smallest numbers entered are displayed.
8. Rewrite the code changing the for loop to while without altering it’s behavior (the output should stay same).

for (let i = 0; i < 3; i++) {

alert( `number ${i}!` );

}

1. Write a loop which prompts for a number greater than 100. If the visitor enters another number – ask him to input again. The loop must ask for a number until either the visitor enters a number greater than 100 or cancels the input/enters an empty line. Here we can assume that the visitor only inputs numbers. There’s no need to implement a special handling for a non-numeric input in this task.
2. An integer number greater than 1 is called a [prime](https://en.wikipedia.org/wiki/Prime_number) if it cannot be divided without a remainder by anything except 1 and itself. In other words, n>1 is a prime if it can’t be evenly divided by anything except 1 and n. For example, 5 is a prime, because it cannot be divided without a remainder by 2, 3 and 4.

**Write the code which outputs prime numbers in the interval from 2 to n.**

For n=10 the result will be 2,3,5,7.

P.S. The code should work for any n, not be hard-tuned for any fixed value.

1. **let z=12;**

**let x;**

for**(x=z;x<=10;x++){**

**x=x\*2;**

}

1. What is the initial value of the index variable?

12

1. What is its end value?

12

1. How many iterations does the loop go through?

0

d. What is the final value of z?

12

1. **let a=10;**

**let b=100;**

**let c=10;**

**let y=0;**

for**(**let **x=a;x<=b;x+=c){**

**y = x \* 2;**

**}**

How many iterations are there?

10

By how much is the index/counter variable incrementing?

10

What is the value of y during its third iteration?

60

What is the final value of y?

200

1. for(let x=10;x>=1;x--){

alert(Convert.toString(x));

}

alert(“Blast off!”);

How many alerts will appear? 11

What will the first alert say? 10

What will the last alert say? Blast off!

What will happen following the last alert? nothing

1. for(let i=2;i<=12;i+=3){

}

How many iterations? 4

What is the last value of i? 14

1. for(let i=4;i>=1;i--){

}

How many iterations? 4

What is the last value of i? 0

1. let x=0;

let i=0;

for(i=1;i<=10;i+=2){

x=x+i;

}

What is the last value of x? 25

What is the last value of i? 11

(loops.js)

1. Write a for loop to display all even numbers from 4 to 140 in alert boxes consecutively.
2. Write a for loop that iterates 5 times, incrementing by 2. Display the index variable in alert boxes.
3. Write a for loop that starts at 10 and ends at 20 and whose body takes the value of the counter/index variables, multiplies it by 2 and assigns this value to a variable y. The body then displays the results in a alert box.
4. Create an application that adds all the numbers between any given range of numbers.
5. Create a program that sums all numbers from 1 to any given number and then finds the average of those numbers.
6. Write a JS program that asks you for a number then prints out the first 1000 terms of that number’s times table. You can use document.write(“html code”); to print to a web page.
7. The Fibonacci sequence works by each term being the sum of the previous two terms added together.

E.g. 0,1,1,2,3,5,8,13,21…

Write a JS program that displays the first 20 numbers of the Fibonacci sequence.