Loops: while and for

We often need to repeat actions.

For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10.

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

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 three iterations.

If `i++` was missing from the example above, the loop would repeat (in theory) forever. In practice, the browser provides ways to stop such loops, and in server-side JavaScript, we can kill the process.

Any expression or variable can be a loop condition, not just comparisons: the condition is evaluated and converted to a boolean by `while`.

For instance, a shorter way to write `while (i != 0)` is `while (i)`:

```
let i = 3;
while (i) { // when i becomes 0, the condition becomes falsy, and the loop stops
  alert( i );
  i--;
}
```

Brackets are not required for a single-line body

If the loop body has a single statement, we can omit the brackets {…}:

```
let i = 3;
while (i) alert(i--);
```

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 should only be used 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

The for loop is the most commonly used loop.

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 false, 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, it could help to go back to the example and reproduce how it runs step-by-step on a piece of paper.

Here’s exactly what 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. This is called an “inline” variable declaration. Such variables are 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 could 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

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++ );
}
```

This makes the loop identical to `while (i < 3)`.

We can actually remove everything, creating an infinite loop:

```
for (;;) {
  // repeats without limits
}
```

Please note that the two `for` semicolons `;` must be present. Otherwise, there would be a syntax error.

Breaking the loop

Normally, a loop exits when its condition becomes falsy.

But we can force the exit at any time using the special `break` directive.

For example, the loop below asks the user for a series of numbers, “breaking” when no number is 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 at the line `(*)` if the user enters an empty line or cancels the input. It stops the loop immediately, passing control to the first line after the loop. Namely, `alert`.

The combination “infinite loop + `break` as needed” is great for situations when a loop’s condition must be checked not in the beginning or end of the loop, but in the middle or even in several places of its body.