6b Theory JS API pages 96 - 102 Loops

Nicholas Bruzzese

July 27, 2025

1 Understanding Loops in JavaScript

Loops are a fundamental concept in programming that allow a piece of code to be executed multiple times based on a specific condition. Unlike conditionals, which run code once if a condition is true, loops keep running the code as long as the condition remains true. This is useful for repetitive tasks, such as counting or processing lists of items. In JavaScript, the two main types of loops discussed here are **while loops** and **for loops**. This lecture explains how these loops work, their structure, and their practical uses, including how to avoid common pitfalls like infinite loops.

1.1 While Loops

A **while loop** is the simplest type of loop in JavaScript. It repeatedly executes a block of code as long as a specified condition is true. The structure of a while loop consists of the `while` keyword, followed by a condition in parentheses, and a body of code enclosed in curly braces. The condition is checked before each iteration of the loop. If the condition is true, the code in the body runs; if it becomes false, the loop stops, and the program moves on to the next piece of code.

For example, consider a scenario where you want to count sheep to help you fall asleep. You can use a while loop to print a message for each sheep counted until you reach a certain number, say 10. The code might look like this:

```
    var sheepCount = 0;
    while (sheepCount < 10) {
      console.log("I have counted " + sheepCount + " sheep!");
      sheepCount++;
    }
    console.log("Zzzzzzzzzzz");

```

Here, a variable `sheepCount` starts at 0. The condition `sheepCount < 10` is checked before each loop. If true, the message is printed, and `sheepCount` is increased by 1 using `sheepCount++`. This repeats until `sheepCount` reaches 10, at which point the condition becomes false (since 10 is not less than 10), and the loop stops. The final output is a sleep message: “Zzzzzzzzzzz”. The output would look like:

```
I have counted 0 sheep!
I have counted 1 sheep!
```

```
I have counted 2 sheep!
...
I have counted 9 sheep!
Zzzzzzzzzzz
```

A critical aspect of while loops is ensuring the condition will eventually become false. If the condition never changes, the loop will run indefinitely, creating an **infinite loop**. For instance, if you forget to include `sheepCount++` in the example above, `sheepCount` remains 0, and the condition `sheepCount < 10` stays true forever, causing the program to print "I have counted 0 sheep!" endlessly. This can crash a program or browser, so always ensure something in the loop changes the condition to eventually stop it.

1.2 For Loops

A **for loop** is another type of loop that is particularly useful when you know how many times you want the loop to run. It combines three parts—setup, condition, and increment—into a single line, making it more concise than a while loop for certain tasks. The structure of a for loop looks like this:

```
for (setup; condition; increment) {
  // Code to run
}

```

- **Setup**: Runs once before the loop starts, usually to create a variable (e.g., `var i = 0`).
- **Condition**: Checked before each iteration. If true, the loop body runs; if false, the loop stops.
- **Increment**: Runs after each iteration to update the loop variable (e.g., `i++`).

Using the sheep-counting example, a for loop could be written as:

```
for (var sheepCount = 0; sheepCount < 10; sheepCount++) {
  console.log("I have counted " + sheepCount + " sheep!");
}
console.log("Zzzzzzzzzzz");

```

This produces the same output as the while loop but is more compact because the variable creation, condition, and increment are all defined in the loop's first line. The setup (`var sheepCount = 0`) initialises the counter, the condition (`sheepCount < 10`) determines when to stop, and the increment (`sheepCount++`) updates the counter after each iteration.

For loops are often used when you want to repeat an action a specific number of times. For example, to print "hello!" three times, you could write:

```
var timesToSayHello = 3;
for (var i = 0; i < timesToSayHello; i++) {
  console.log("hello!");
}

```

This outputs:

```
hello!
hello!
hello!
```

Here, the variable `i` starts at 0 and increases by 1 each time until it reaches 3, at which point the condition `i < timesToSayHello` becomes false, and the loop stops.

1.3 Using For Loops with Arrays and Strings

For loops are especially powerful when working with **arrays** (lists of items) or **strings** (sequences of characters). Arrays and strings in JavaScript can be accessed using **indexes**, which are numbers starting from 0 that indicate the position of an item or character. A for loop can iterate through each index to access every element or character.

For example, to print the names of animals in an array representing a zoo, you could use:

```
  var animals = ["Lion", "Flamingo", "Polar Bear", "Boa Constrictor"];
  for (var i = 0; i < animals.length; i++) {
    console.log("This zoo contains a " + animals[i] + ".");
  }

```

The property `animals.length` gives the number of items in the array (4 in this case). The loop runs from `i = 0` to `i < 4`, accessing each animal using `animals[i]`. The output is:

```
This zoo contains a Lion.
This zoo contains a Flamingo.
This zoo contains a Polar Bear.
This zoo contains a Boa Constrictor.
```

Similarly, you can use a for loop to process each character in a string. For example:

```
  var name = "Nick";
  for (var i = 0; i < name.length; i++) {
    console.log("My name contains the letter " + name[i] + ".");
  }

```

This outputs:

```
My name contains the letter N.
My name contains the letter i.
My name contains the letter c.
My name contains the letter k.
```

Here, `name.length` is 4, so the loop runs from `i = 0` to `i < 4`, accessing each character with `name[i]`.

1.4 Other Ways to Use For Loops

For loops are flexible and don't always need to start at 0 or increment by 1. You can adjust the setup, condition, and increment to suit different tasks. For example, to print powers of 2 less than 10,000, you could write:

```javascript
  for (var x = 2; x < 10000; x = x * 2) {
    console.log(x);
  }

```

This loop starts with `x = 2`, checks if `x < 10000`, and doubles `x` each time using `x = x * 2`. The output is:

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
```

This shows how for loops can handle more complex patterns by changing how the loop variable is updated.

2 Summary

Loops in JavaScript, specifically **while loops** and **for loops**, allow you to repeat code multiple times based on a condition. While loops are simple and run until a condition becomes false, but you must ensure the condition eventually changes to avoid infinite loops. For loops are more structured, combining setup, condition, and increment, making them ideal for tasks with a known number of repetitions, such as iterating over arrays or strings. By using loops, you can efficiently handle repetitive tasks like counting, processing lists, or generating sequences, making your programs more powerful and flexible. Always test your loops to ensure they stop as expected and produce the correct output.