The for and for-in loops

In this lesson, you will have an overview of all flow-control statements provided by the JavaScript language.

WE'LL COVER THE FOLLOWING

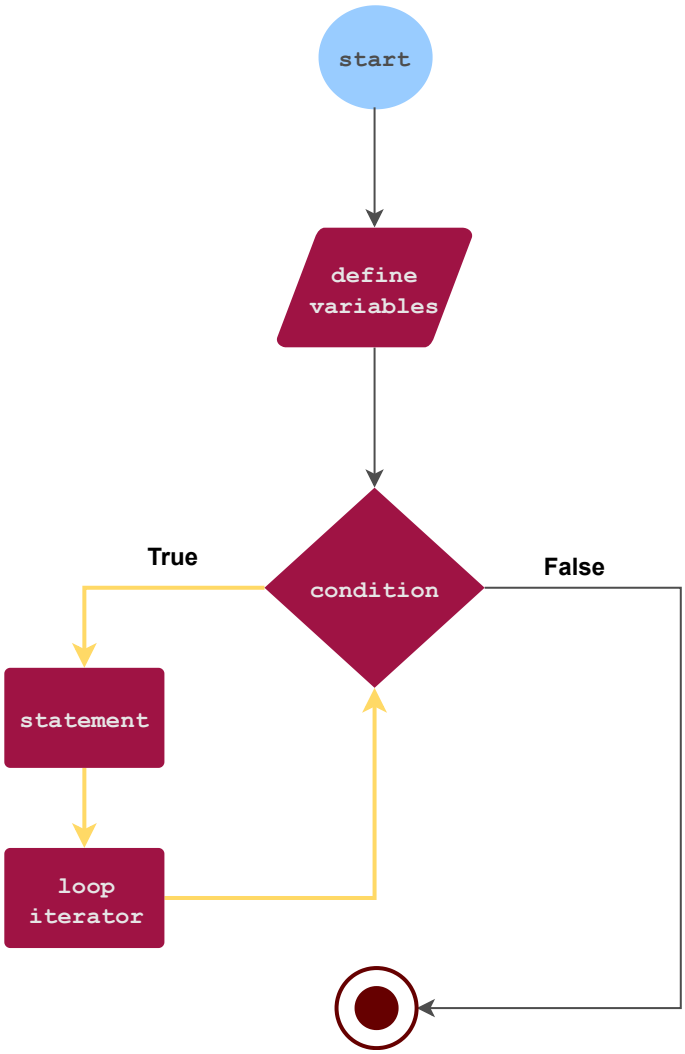
The `for` Loop

Just as in many programming languages, the for loop provides a pretest with

optional variable initialization and optional post-loop code to be executed:

Illustration

Here is an illustration to explain the above concept



Syntax

The syntax of the `for` loop in JavaScript is as follows:

```
for (initialization ; condition ; post_loop_expression) statement
```

for loop statement syntax in JavaScript

Example

This example shows the usage of a `for` loop:

```
for (var i = 0; i < 10; i++) {
    console.log(i);
}
```

The `initialization`, condition, and post_loop_expression are all optional, so you can create an infinite loop by omitting all of them:

```
for (;;;) {
  console.log("Nothing gonna stop me");
}
```

The `for-in` loop

There is another for loop in JavaScript that can be used only to enumerate the properties of an object, the `for-in` loop:

```
for (property in expression) statement
```

for-in loop statement syntax in JavaScript

The expression should retrieve on `object`. The loop iterates through all object properties and retrieves the current property name in property.

Examples

Here is an example that retrieves all properties of the document object:

```
for (var propName in document) {
    console.log(propName);
}
```

> 📜**NOTE:** The `for-in` statement will throw an error if the variable representing the object to iterate over is null or undefined. The ECMAScript standard does not specify the order of the properties retrieved, so it may be different depending on the browser that runs the script.

The `break` and `continue` statements

You have seen that JavaScript implements a number of loops, such as `while`, `do-while`, `for`, and `for-in`.

Two statements, `break` and `continue`, provide more control upon loops.

The break statement immediately aborts the loop and passes the execution to the next statement following the loop.

On the other hand, the continue statement exits the current loop but execution continues from the top of the loop.

Example

Let's see an example for `break`:

```
var nums = [12, 42, 23, 2, 6, 17, 21]
var index = -1;
for (var i = 0; i < nums.length; i++) {
    if (nums[i] == 6) {
        index = i;
        break;
    }
}
console.log(index);
```

The `for` loop in this code snippet traverses the `nums` array and searches for the element that contains `6`.

As soon as this element is found, its index is saved and the `for` loop is terminated by `break`.

Example

This example demonstrates the usage of `continue`:

```
var nums = [12, 42, 23, 2, 6, 17, 21]
for (var i = 0; i < nums.length; i++) {
    if (nums[i] % 2 == 0) continue;
    nums[i] *= 2.5;
    console.log("nums[" + i + "] = " + nums[i]);
}
```

The code iterates through the elements of the nums array and multiplies all

elements with 2.5, which represent odd numbers.

Here the continue statement is used to terminate the current loop when an even number is found. This technique is often used to decrease the nesting of a code block; though, if and continue can be replaced by an if statement with the inverted condition.

Both statements have a form when a label is specified. In this case break and continue return a particular location in the code.

Any statement can be labeled with the following syntax:

```
label_name: statement
```

label name syntax in JavaScript

Example

Take a look at this example:

```
var counter = 0;
outer_loop:
    for (var i = 0; i < 10; i++) {
        for (var j = 0; j < 10; j++) {
            if (i == 7 && j == 7) {
                continue outer_loop;
            }
            if (i == 8 && j == 4) {
                break outer_loop;
            }
            counter++;
        }
    }
console.log(counter); // 81
```

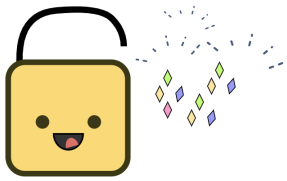
This code contains two nested for loops, each executing its body ten times, and the internal loop increments counter. Without interrupting the loops, the counter should be 100 at the end. However, the inner loop is interrupted twice. First, when `i` equals 7 and `j` equals 7. Continue outer_loop is used, so the execution goes on with the outer for loop.

It means that three internal loops are omitted. The second time is when `i`

equals 8 and `j` equals 4, break outer_loop aborts not only the internal, but the outer loop, too.

It means that six internal loops (it would increment counter by six) and an outer loop (it would increment counter by ten) is omitted. So, 19 increments are omitted in total, and that is why counter is set to 81 (100–19).

Achievement unlocked! 🎉



Congratulations! You've learned how to use the `for` and `for-in` loops in JavaScript.

Great work! Give yourself a round of applause! :)

In the *next lesson*, we will study the `switch` and `with` statements.

See you there! :)