Extra notes - JavaScript - Logic - Part 1

- Dr Nick Hayward

A brief introduction to logic in JavaScript.

Contents

Intro JavaScript is now a core, invaluable technology for client-side design and development. From plain JavaScript to the latest library, its growth as a development environment has exploded over the last few years. It is now being used as a powerful technology to help us rapidly prototype and develop web, mobile, and desktop applications. We can also use it with embedded systems.

Logic A few underlying concepts for working with *logic* in JavaScript.

blocks A natural coding style, for JS and other languages, is the simple act of grouping contiguous and related code statements together. Often known as **blocks**, in JS a block is defined by wrapping one or more statements together within a pair of curly braces, `{}` .

Such **blocks** are commonly attached to other forms of control statement, including conditional statements,

```
if (a > b) {
...do something useful...
}
```

conditionals Conditionals, and by association conditional statements, inherently require a decision to be made. A code statement, and application, will often need to consult **state** and the answer will predominantly be a simple **yes** or **no**.

Within our JS applications, there are many different ways we can express **conditionals**. The most common example is the `if` statement. In essence, we use this statement to check, *if this given condition is true, do the following…*

```
if (a > b) {
console.log("a is greater than b...");
}
```

The `if` statement requires an expression between the parentheses that can be treated as either *true* or *false*.

We can add an additional option if this expression returns false, using a common `else` clause

```
if (a > b) {
console.log("a is greater than b...");
} else {
console.log("no, b is greater...");
}
```

As mentioned above, types that are not matching, in effect the expected type for the comparison, will be coerced by JS to the expected type. For an `if` statement, JS expects a `boolean` .

With this in mind, JS defines a list of values that it considers *false*. These values will become false when coerced to a `boolean` . For example, such values include `0` (and `""`). This means that any value not on this list of *false* values will be considered true, and therefore coerced to *true* when defined as a `boolean`
.

Conditionals in JS also exist in another form, which includes the `switch` statement. Further details on conditionals later on.

loops Programming in general, and JS in this instance, uses loops to allow repeating sets of actions until a given condition fails. In effect, this repetition continues whilst the requested condition holds.

Loops can take many different forms, but in essence they follow this basic behaviour.

A loop includes the *test condition* as well as a *block*, normally within curly braces. Each time this block executes, an iteration of the loop has occurred.

Good examples of this behaviour include the `while` and `do...while` loops. Each repeat a block of statements until a condition ceases to evaluate as `true` .

The basic difference between these loops, `while` and `do...while` , is whether the conditional tested is before the first iteration (`while` loop), or after the first iteration (`do...while`) loop.

If the conditional test returns as `false` , the next iteration of both of these loops will fail to execute. The loop stops.

So, if the condition is initially false, a `while` loop will never run, but a `do...while` will run through for the first time.

We can also stop a JS loop using the common `break` statement.

Another useful form of loop is known as the `for` loop. This loop has three clauses, including

- initialisation clause
- conditional test clause
- update clause

If the goal of the loop is counting, or iterating over a large list or array, it is often more efficient to use a `for` loop. It will often also be the easier option.

There are other specialised forms of loop that will be covered later on.

n.b. don't forget, programming languages, and Computer Science in general, start counting at `0` .

References

- MDN
  - MDN - JS
  - MDN - JS Const
  - MDN - JS Data Types and Data Structures
  - MDN - JS Grammar and Types
- W3 - JS Performance