JS follows the order the operations

The following are some operators

+

-

\*

/

%

Binary operators take two values (+, -, \*, &&, etc,)

Unary operators take one (-, typeof, !)

// minus can be used as both

Special Numbers

Infinity

-Infinity

Nan (not a number)

strings

Strings can be marked with quotes “word”, ‘word’, `word`.

\t to tab

Note: javaScript’s string element uses 16 bits, 2^16 different characters, Unicode has more than that. So, some characters, like emoji, take up two character positions, more later.

Using ‘`’, back-quoted strings, called “template literals”, can embed values in them

`hello ${name}`

Note: JavaScript also implicitly parses strings to ints based on Unicode, so (“A” < “B”) would be true.

When comparing string JS goes from left to right for each character so (“aG” < “aZ”) would be true.

== will compare two values, ignoring their type

=== will compare two values, and their type (as a beginner, this is best to always use)

localeCompare(string) is like strcmp(string) from C, returning -1,0 or 1 //fuck that shit

NaN == NaN is false

//NaN denotes a nonsensical computation, as such isn’t equal to another nonsensical computation

Logical operators

&&, ||, !

Ternary operators (3 values)

isTrue ? ifTrueValue : ifFalseValue

**Empty values**

There are two special values, written null and undefined, that are used to denote the absence of a *meaningful* value. They are themselves values, but they carry no information.

Many operations in the language that don’t produce a meaningful value (you’ll see some later) yield undefined simply because they have to yield *some* value.

The difference in meaning between undefined and null is an accident of JavaScript’s design, and it doesn’t matter most of the time. In cases where you actually have to concern yourself with these values, I recommend treating them as mostly interchangeable.

Short-Circuting of Logical Operators

&& and || handle values of different types in a peculiar way.

They will convert the left value to a Bool.

Depending on the Bool conversion result

And the operator.

Will either return the original Left or right value

Example for ||

console.log (null || “unnamed”)

// -> unnamed

console.log(“lou” || “unnamed”)

// -> unnamed

For &&, when the left is false it returns that value, otherwise it returns the value on the right

console.log (7 && 8)

// -> 8

console.log (null && 8)

//->

//one possible use case

Another important thing is the part to the right is evaluated only when necessary.

In true || function() and false && function(), function() is never going to run. This called short-circuit evaluation

Some great short-circuit use cases <https://stackoverflow.com/questions/2580136/does-python-support-short-circuiting>

The same can be done with comparison and ternary operators