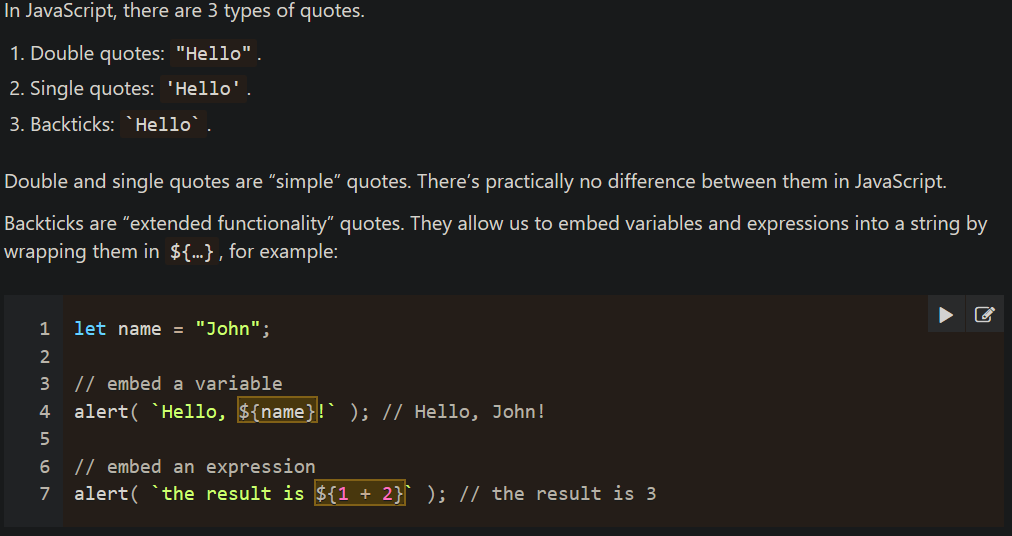
# The modern mode, "use strict"

## [“use strict”](https://javascript.info/strict-mode" \l "use-strict)

The directive looks like a string: "use strict" or 'use strict'. When it is located at the top of a script, the whole script works the “modern” way.

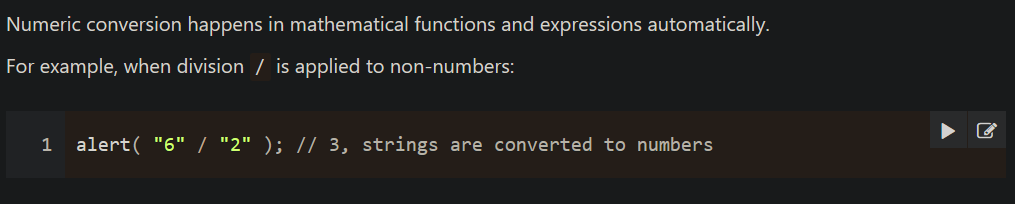
# Data types

## [String](https://javascript.info/types" \l "string)



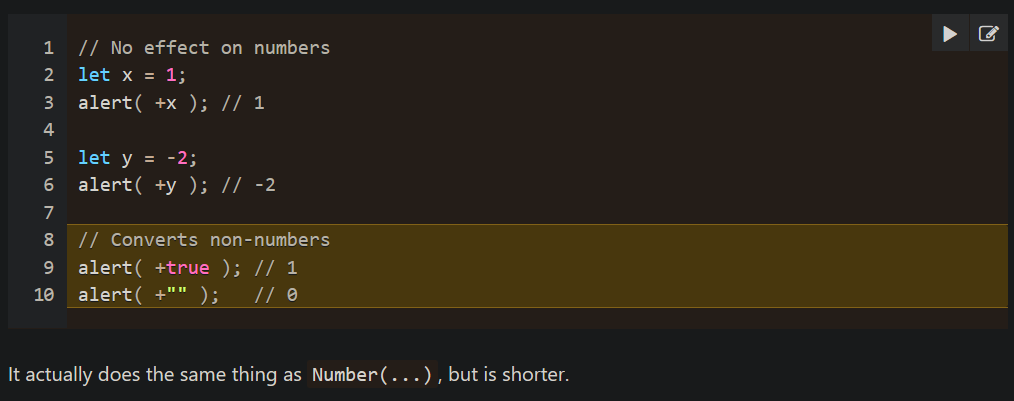
# Type Conversions

## [Numeric Conversion](https://javascript.info/type-conversions" \l "numeric-conversion)

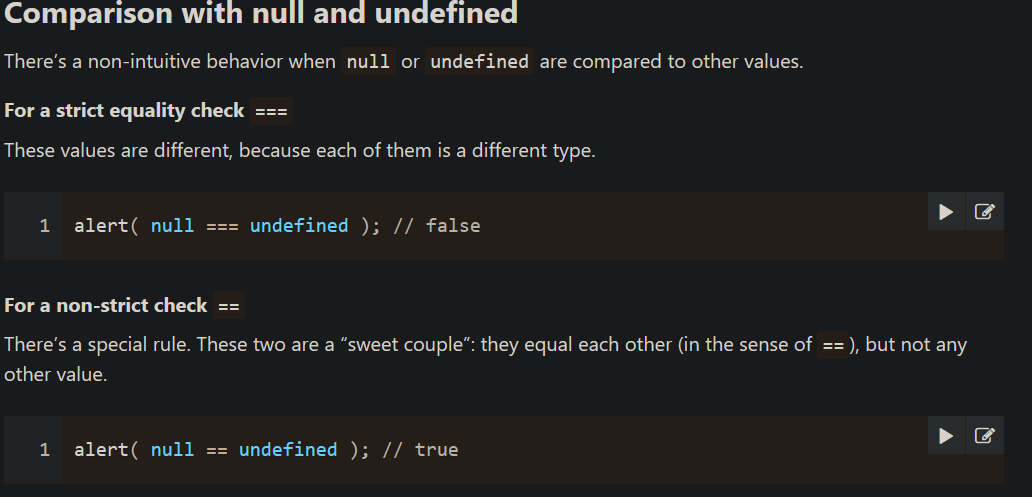


# Operators

## [Numeric conversion, unary +](https://javascript.info/operators" \l "numeric-conversion-unary)

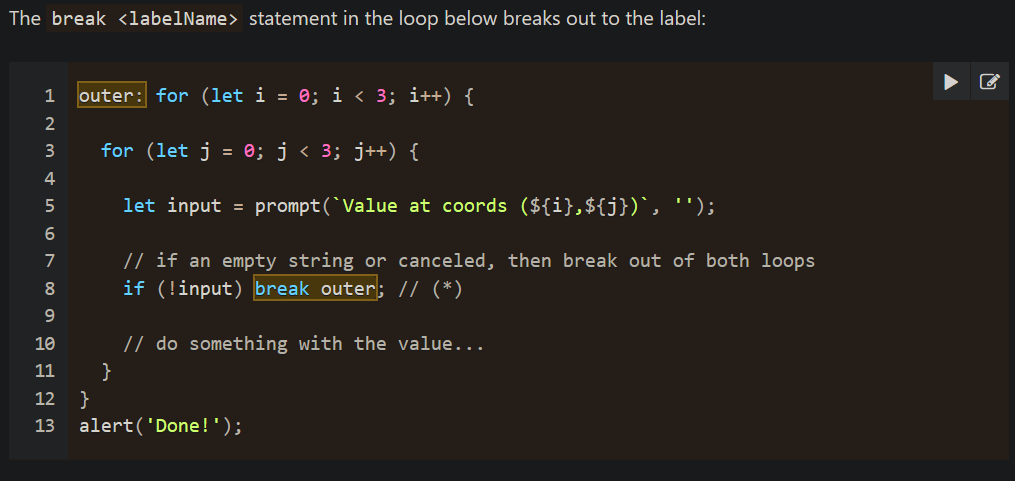


# Comparisons

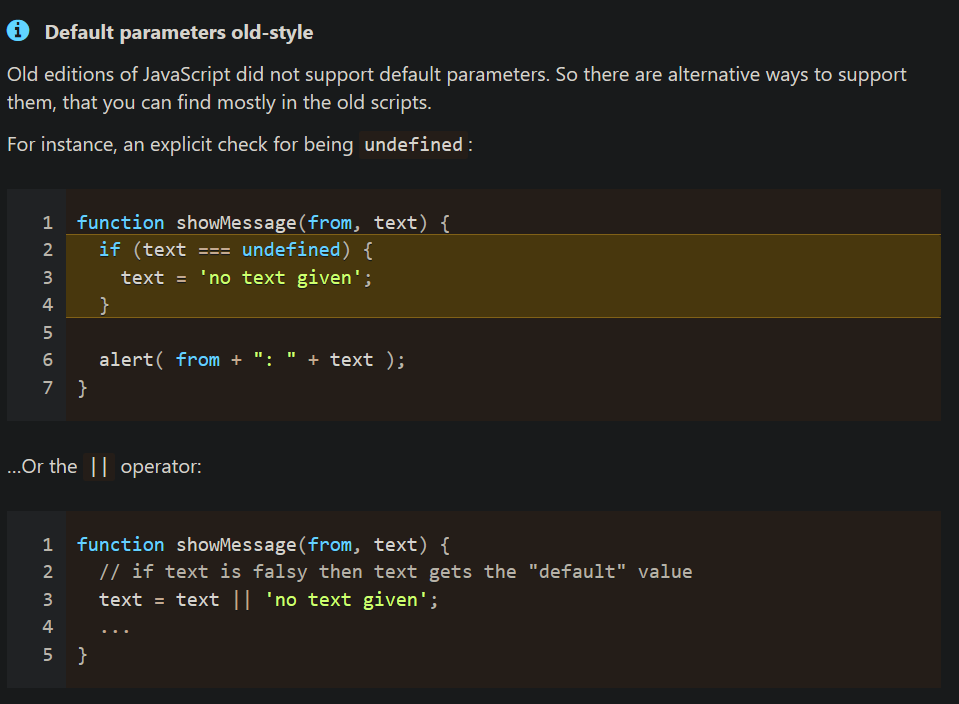


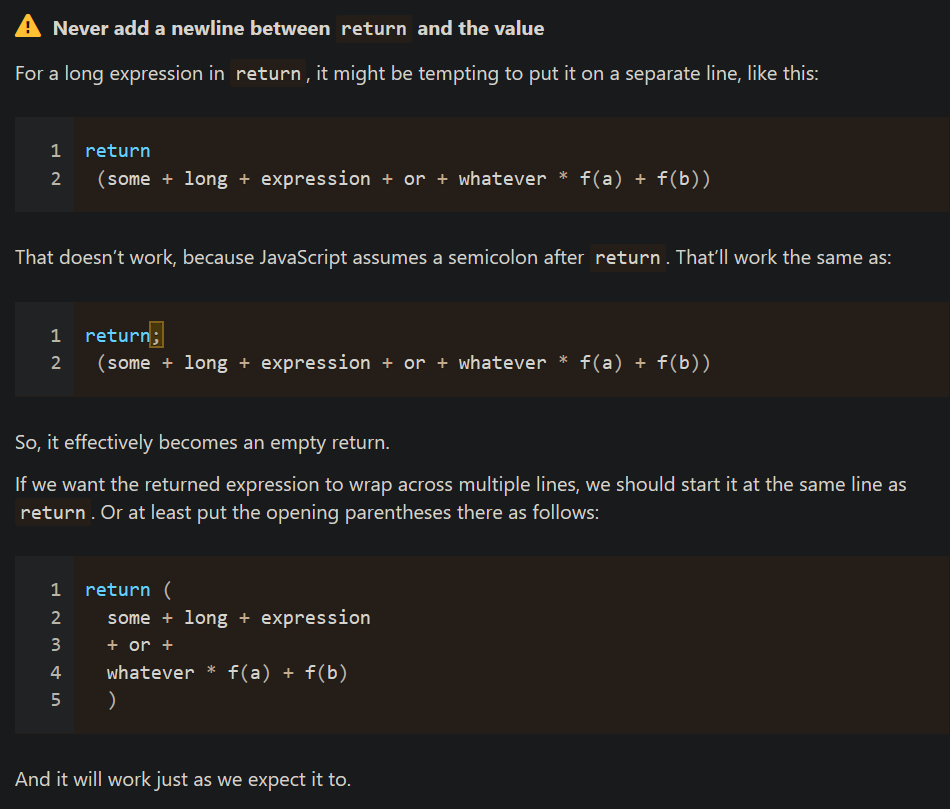
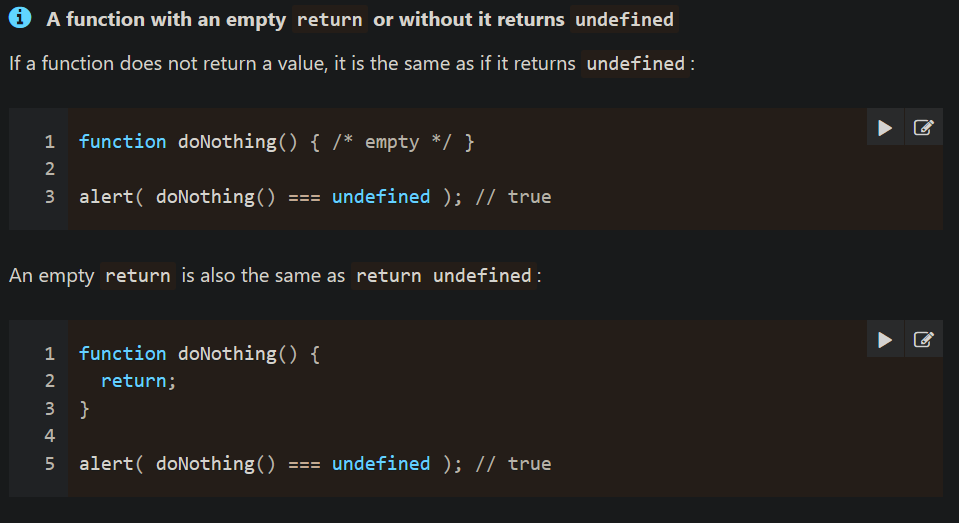
# Loops: while and for

[**Labels for break/continue**](https://javascript.info/while-for#labels-for-break-continue)

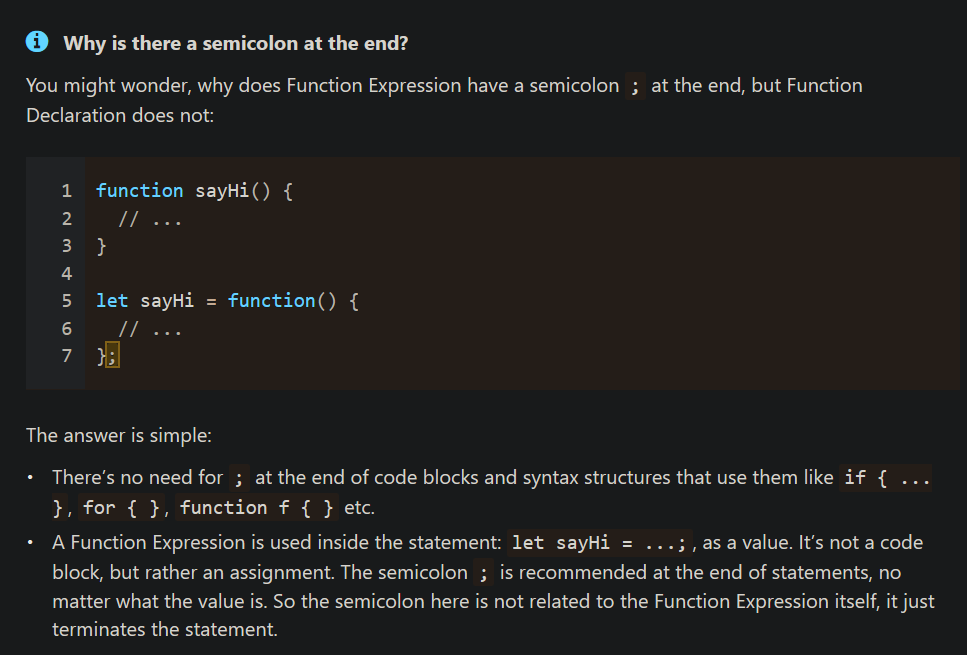


Functions Basics





# Function expressions



## [Function Expression vs Function Declaration](https://javascript.info/function-expressions" \l "function-expression-vs-function-declaration)

1. **Created by the JavaScript engine.**
   1. **A Function Expression is created when the execution reaches it and is usable only from that moment.**
   2. **A Function Declaration can be called earlier than it is defined.**
2. **Block scope.**
   1. **In strict mode, when a Function Declaration is within a code block, it’s visible everywhere inside that block. But not outside of it.**

# Objects

## [Property names limitations](https://javascript.info/object" \l "property-names-limitations)

Property names (keys) must be either strings or symbols (a special type for identifiers, to be covered later).

Other types are automatically converted to strings.

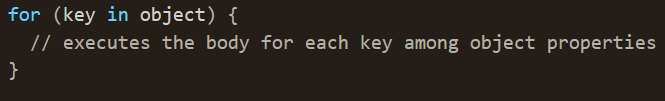
For instance, a number 0 becomes a string "0" when used as a property key

## [Square brackets](https://javascript.info/object" \l "square-brackets)

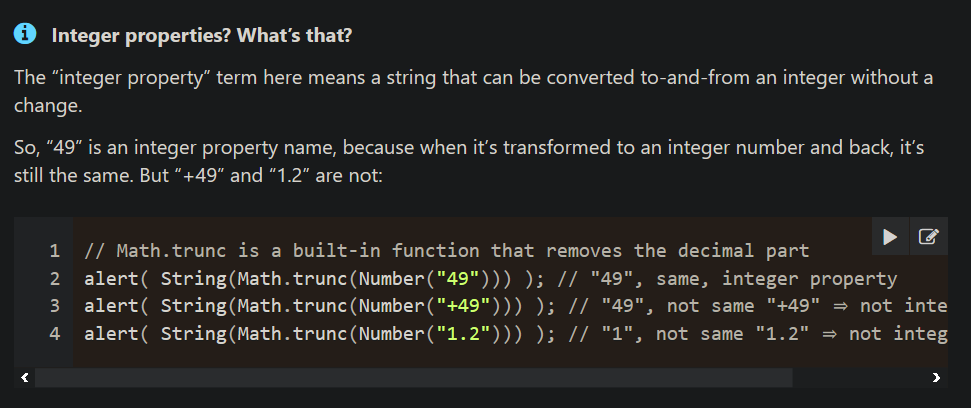
## [Property value shorthand](https://javascript.info/object" \l "property-value-shorthand)[Property existence test, “in” operator](https://javascript.info/object" \l "property-existence-test-in-operator)

## 

## [The “for…in” loop](https://javascript.info/object" \l "the-for-in-loop)

****

### [Ordered like an object](https://javascript.info/object" \l "ordered-like-an-object)

The short answer is: “ordered in a special fashion”: integer properties are sorted, others appear in creation order. We can “cheat” by making the codes non-integer. Adding a plus "+" sign before each code is enough. To get number I Creation order.

## [Copying by reference](https://javascript.info/object" \l "copying-by-reference)

## **When an object variable is copied – the reference is copied, the object is not duplicated.** [Cloning and merging, Object.assign](https://javascript.info/object" \l "cloning-and-merging-object-assign)

# [Symbol type](https://javascript.info/symbol) (IMPORTANT)

# [Object to primitive conversion](https://javascript.info/object-toprimitive) (Need Revision)

# Numbers

## [toString(base)](https://javascript.info/number" \l "tostring-base)

The method num.toString(base) returns a string representation of num in the numeral system with the given base.

## [Tests: isFinite and isNaN](https://javascript.info/number" \l "tests-isfinite-and-isnan)

## [parseInt and parseFloat](https://javascript.info/number" \l "parseint-and-parsefloat)

Numeric conversion using a plus + or Number() is strict. If a value is not exactly a number, it fails: The sole exception is spaces at the beginning or at the end of the string, as they are ignored.

But in real life we often have values in units, like "100px" or "12pt" in CSS. Also in many countries the currency symbol goes after the amount, so we have "19€" and would like to extract a numeric value out of that.

[**for..in v/s for..of v/s ForEach**](https://hackernoon.com/3-javascript-performance-mistakes-you-should-stop-doing-ebf84b9de951)

# Array methods

## [Searching in array](https://javascript.info/array-methods" \l "searching-in-array)

## [Transform an array](https://javascript.info/array-methods" \l "transform-an-array)

# [Example](https://javascript.info/task/calculator-extendable)[Iterables](https://javascript.info/iterable)(Need revision)

# Map and Set

## 

## [Iteration over Map](https://javascript.info/map-set#iteration-over-map)

## [Object.entries: Map from Object](https://javascript.info/map-set" \l "object-entries-map-from-object)

## [Object.fromEntries: Object from Map](https://javascript.info/map-set" \l "object-fromentries-object-from-map)

## [Set](https://javascript.info/map-set" \l "set)

# WeakMap and WeakSet

* Usually, properties of an object or elements of an array or another data structure are considered reachable and kept in memory while that data structure is in memory.
* For instance, if we put an object into an array, then while the array is alive, the object will be alive as well, even if there are no other references to it.
* Similar to that, if we use an object as the key in a regular Map, then while the Map exists, that object exists as well. It occupies memory and may not be garbage collected.

## [WeakMap](https://javascript.info/weakmap-weakset" \l "weakmap)

## The first difference from Map is that WeakMap keys must be objects, not primitive values: [WeakSet](https://javascript.info/weakmap-weakset" \l "weakset)

# Destructuring assignment

### [The rest ‘…’](https://javascript.info/destructuring-assignment" \l "the-rest)

### [Default values](https://javascript.info/destructuring-assignment" \l "default-values)

## [Object destructuring](https://javascript.info/destructuring-assignment" \l "object-destructuring)(Need revision)

## [Nested destructuring](https://javascript.info/destructuring-assignment" \l "nested-destructuring)

## 

# Rest parameters and spread syntax

## [Spread syntax](https://javascript.info/rest-parameters-spread#spread-syntax)[Lexical Environment](https://javascript.info/closure" \l "lexical-environment)(Need revision)

# Closure(Important)

# Global object(Need Revision)

# Function object, NFE