jq manual (excerpt)

Conditionals and Comparisons

==, !=

The expression ‘a == b’ will produce ‘true’ if the results of evaluating a and b are equal (that is, if they represent equivalent JSON values) and ‘false’ otherwise. In particular, strings are never considered equal to numbers. In checking for the equality of JSON objects, the ordering of keys is irrelevant. If you’re coming from JavaScript, please note that jq’s == is like JavaScript’s ===, the “strict equality” operator.

!= is “not equal”, and ‘a != b’ returns the opposite value of ‘a == b’

Examples

Filter	`. == false`
Input	`null`
Output	`false`

Filter	`. == {"b": {"d": (4 + 1e-20), "c": 3}, "a":1}`
Input	`{"a":1, "b": {"c": 3, "d": 4}}`
Output	`true`

Filter	`.[] == 1`
Input	`[1, 1.0, "1", "banana"]`
Output	`true`
	`true`
	`false`
	`false`

if-then-else-end

`if A then B else C end` will act the same as `B` if `A` produces a value other than false or null, but act the same as `C` otherwise.

`if A then B end` is the same as `if A then B else . end`. That is, the `else`

branch is optional, and if absent is the same as `.`. This also applies to `elif` with absent ending `else` branch.

Checking for false or null is a simpler notion of "truthiness" than is found in JavaScript or Python, but it means that you'll sometimes have to be more explicit about the condition you want. You can't test whether, e.g. a string is empty using `if .name then A else B end`; you'll need something like `if .name == "" then A else B end` instead.

If the condition `A` produces multiple results, then `B` is evaluated once for each result that is not false or null, and `C` is evaluated once for each false or null.

More cases can be added to an if using `elif A then B` syntax.

Examples

Filter	`if . == 0 then "zero" elif . == 1 then "one" else "many" end`
Input	`2`
Output	`"many"`

>, >=, <=, <

The comparison operators >, >=, <=, < return whether their left argument is greater than, greater than or equal to, less than or equal to or less than their right argument (respectively).

The ordering is the same as that described for `sort`, above.

Examples

Filter	`. < 5`
Input	`2`
Output	`true`

and, or, not

jq supports the normal Boolean operators `and`, `or`, `not`. They have the same standard of truth as if expressions - `false` and `null` are considered "false values", and anything else is a "true value".

If an operand of one of these operators produces multiple results, the operator itself will produce a result for each input.

`not` is in fact a builtin function rather than an operator, so it is called as a filter to which things can be piped rather than with special syntax, as in `.foo and .bar | not`.

These three only produce the values `true` and `false`, and so are only useful for genuine Boolean operations, rather than the common Perl/Python/Ruby idiom of "value_that_may_be_null or default". If you want to use this form of "or", picking between two values rather than evaluating a condition, see the `//` operator below.

Examples

Filter	`42 and "a string"`
Input	`null`
Output	`true`

Filter	`(true, false) or false`
Input	`null`
Output	`true`
	`false`

Filter	`(true, true) and (true, false)`
Input	`null`
Output	`true`
	`false`
	`true`
	`false`

Filter	`[true, false | not]`
Input	`null`
Output	`[false, true]`