|  |  |
| --- | --- |
| seq {base} | R Documentation |

**Sequence Generation**

**Description**

Generate regular sequences. seq is a standard generic with a default method. seq.int is a primitive which can be much faster but has a few restrictions. seq\_along and seq\_len are very fast primitives for two common cases.

**Usage**

seq(...)

## Default S3 method:

seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),

length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq\_along(along.with)

seq\_len(length.out)

**Arguments**

|  |  |
| --- | --- |
| ... | arguments passed to or from methods. |
| from, to | the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument. |
| by | number: increment of the sequence. |
| length.out | desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional. |
| along.with | take the length from the length of this argument. |

**Details**

Numerical inputs should all be [finite](http://127.0.0.1:14695/library/base/help/finite) (that is, not infinite, [NaN](http://127.0.0.1:14695/library/base/help/NaN) or NA).

The interpretation of the unnamed arguments of seq and seq.int is *not* standard, and it is recommended always to name the arguments when programming.

seq is generic, and only the default method is described here. Note that it dispatches on the class of the **first** argument irrespective of argument names. This can have unintended consequences if it is called with just one argument intending this to be taken as along.with: it is much better to use seg\_along in that case.

seq.int is an [internal generic](http://127.0.0.1:14695/library/base/help/internal%20generic) which dispatches on methods for "seq" based on the class of the first supplied argument (before argument matching).

Typical usages are

seq(from, to)

seq(from, to, by= )

seq(from, to, length.out= )

seq(along.with= )

seq(from)

seq(length.out= )

The first form generates the sequence from, from+/-1, ..., to (identical to from:to).

The second form generates from, from+by, ..., up to the sequence value less than or equal to to. Specifying to - from and by of opposite signs is an error. Note that the computed final value can go just beyond to to allow for rounding error, but is truncated to to. (‘Just beyond’ is by up to *1e-10* times abs(from - to).)

The third generates a sequence of length.out equally spaced values from from to to. (length.out is usually abbreviated to length or len, and seq\_len is much faster.)

The fourth form generates the integer sequence 1, 2, ..., length(along.with). (along.with is usually abbreviated to along, and seq\_along is much faster.)

The fifth form generates the sequence 1, 2, ..., length(from) (as if argument along.with had been specified), *unless* the argument is numeric of length 1 when it is interpreted as 1:from (even for seq(0) for compatibility with S). Using either seq\_along or seq\_len is much preferred (unless strict S compatibility is essential).

The final form generates the integer sequence 1, 2, ..., length.out unless length.out = 0, when it generates integer(0).

Very small sequences (with from - to of the order of *10^{-14}* times the larger of the ends) will return from.

For seq (only), up to two of from, to and by can be supplied as complex values provided length.out or along.with is specified. More generally, the default method of seq will handle classed objects with methods for the Math, Ops and Summary group generics.

seq.int, seq\_along and seq\_len are [primitive](http://127.0.0.1:14695/library/base/help/primitive).

**Value**

seq.int and the default method of seq for numeric arguments return a vector of type "integer" or "double": programmers should not rely on which.

seq\_along and seq\_len return an integer vector, unless it is a *long vector* when it will be double.

**References**

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

**See Also**

The methods [seq.Date](http://127.0.0.1:14695/library/base/help/seq.Date) and [seq.POSIXt](http://127.0.0.1:14695/library/base/help/seq.POSIXt).

[:](http://127.0.0.1:14695/library/base/help/:), [rep](http://127.0.0.1:14695/library/base/help/rep), [sequence](http://127.0.0.1:14695/library/base/help/sequence), [row](http://127.0.0.1:14695/library/base/help/row), [col](http://127.0.0.1:14695/library/base/help/col).

**Examples**

seq(0, 1, length.out = 11)

seq(stats::rnorm(20)) # effectively 'along'

seq(1, 9, by = 2) # matches 'end'

seq(1, 9, by = pi) # stays below 'end'

seq(1, 6, by = 3)

seq(1.575, 5.125, by = 0.05)

seq(17) # same as 1:17, or even better seq\_len(17)