Package ‘assertive.numbers’

August 29, 2016

Type Package

Title Assertions to Check Properties of Numbers

Version 0.0-2

Date 2016-05-05

Author Richard Cotton [aut, cre]

Maintainer Richard Cotton <richierocks@gmail.com>

Description A set of predicates and assertions for checking the properties of numbers. This is mainly for use by other package developers who want to include run-time testing features in their own packages. End-users will usually want to use assertive directly.

URL https://bitbucket.org/richierocks/assertive.numbers

BugReports https://bitbucket.org/richierocks/assertive.numbers/issues

Depends R (>= 3.0.0)

Imports assertive.base (>= 0.0-2)

Suggests testthat

License GPL (>= 3)

LazyLoad yes

LazyData yes

Acknowledgments Development of this package was partially funded by the Proteomics Core at Weill Cornell Medical College in Qatar <http://qatar-weill.cornell.edu>. The Core is supported by 'Biomedical Research Program' funds, a program funded by Qatar Foundation.

Collate 'assert-is-divisible-by.R' 'assert-is-equal-to.R' 'imports.R' 'assert-is-in-range.R' 'assert-is-infinity-nan.R' 'assert-is-real-imaginary.R' 'assert-is-whole-number.R' 'is-divisible-by.R' 'is-equal-to.R' 'is-in-range.R' 'is-infinity-nan.R' 'is-real-imaginary.R' 'is-whole-number.R'

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-05-09 21:37:47

R topics documented:

`assert_all_are_divisible_by` *Is the input divisible by a number?*

Description

Checks to see if the input is divisible by some number.

Usage

```
assert_all_are_divisible_by(x, n, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_divisible_by(x, n, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_even(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_even(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_odd(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_odd(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_divisible_by(x, n, tol = 100 * .Machine$double.eps,
  .xname = get_name_in_parent(x))

is_even(x, tol = 100 * .Machine$double.eps, .xname = get_name_in_parent(x))
```

```
is_odd(x, tol = 100 * .Machine$double.eps, .xname = get_name_in_parent(x))
```

Arguments

x	A numeric vector to divide.
n	A numeric vector to divide by.
tol	Differences from zero smaller than tol are not considered.
na_ignore	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like na.rm in many stats package functions, except that the position of the failing values does not change.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

TRUE if the input x is divisible by n, within the specified tolerance.

Note

is_even and is_odd are shortcuts for divisibility by two.

See Also

is_whole_number

Examples

```
is_divisible_by(1:10, 3)
is_divisible_by(-5:5, -2)
is_divisible_by(1.5:10.5, c(1.5, 3.5))
assert_any_are_even(1:10)
assertive.base::dont_stop(assert_all_are_even(1:10))
```

---

assert_all_are_equal_to

How does the input relate to a value?

Description

Is x equal/not equal/greater than/less than y?

Usage

```
assert_all_are_equal_to(x, y, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_equal_to(x, y, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_not_equal_to(x, y, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_not_equal_to(x, y, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_greater_than(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_greater_than(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_greater_than_or_equal_to(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_greater_than_or_equal_to(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_less_than(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_less_than(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_less_than_or_equal_to(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_less_than_or_equal_to(x, y, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_equal_to(x, y, tol = 100 * .Machine$double.eps,
  .xname = get_name_in_parent(x), .yname = get_name_in_parent(x))

is_not_equal_to(x, y, tol = 100 * .Machine$double.eps,
  .xname = get_name_in_parent(x), .yname = get_name_in_parent(x))

is_greater_than(x, y, .xname = get_name_in_parent(x),
  .yname = get_name_in_parent(x))

is_greater_than_or_equal_to(x, y, .xname = get_name_in_parent(x),
  .yname = get_name_in_parent(x))
```

```
is_less_than(x, y, .xname = get_name_in_parent(x),
  .yname = get_name_in_parent(x))

is_less_than_or_equal_to(x, y, .xname = get_name_in_parent(x),
  .yname = get_name_in_parent(x))
```

Arguments

x	A numeric vector.
y	Another numeric vector, typically scalar or the same length as x. See note.
tol	Values within tol are considered equal.
na_ignore	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like na.rm in many stats package functions, except that the position of the failing values does not change.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.
.yname	Not intended to be used directly.

Value

TRUE if the input x is equal/not equal/greater than/less than y

Note

The usual recycling rules apply when x and y are different lengths. See Intro to R for details: https://cran.r-project.org/doc/manuals/r-devel/R-intro.html#Vector-arithmetic https://cran.r-project.org/doc/manuals/r-devel/R-intro.html#The-recycling-rule

Examples

```
# Approximate and exact floating point comparisons:
# See FAQ on R 7.31
x <- sqrt(2) * sqrt(2)
is_equal_to(x, 2)
is_equal_to(x, 2, tol = 0)
is_not_equal_to(x, 2)
is_not_equal_to(x, 2, tol = 0)

# Elements of x and y are recycled
is_equal_to(1:6, 1:3)

# Inequalities
x <- c(1 - .Machine$double.neg.eps, 1, 1 + .Machine$double.eps)
is_greater_than(x, 1)
is_greater_than_or_equal_to(x, 1)
is_less_than(x, 1)
is_less_than_or_equal_to(x, 1)
```

assert_all_are_finite *Are the inputs (in)finite?*

Description

Checks to see if the inputs are (in)finite.

Usage

```
assert_all_are_finite(x, severity = getOption("assertive.severity", "stop"))

assert_any_are_finite(x, severity = getOption("assertive.severity", "stop"))

assert_all_are_infinite(x, severity = getOption("assertive.severity", "stop"))

assert_any_are_infinite(x, severity = getOption("assertive.severity", "stop"))

assert_all_are_negative_infinity(x, severity = getOption("assertive.severity",
  "stop"))

assert_any_are_negative_infinity(x, severity = getOption("assertive.severity",
  "stop"))

assert_all_are_positive_infinity(x, severity = getOption("assertive.severity",
  "stop"))

assert_any_are_positive_infinity(x, severity = getOption("assertive.severity",
  "stop"))

is_finite(x, .xname = get_name_in_parent(x))

is_infinite(x, .xname = get_name_in_parent(x))

is_negative_infinity(x, .xname = get_name_in_parent(x))

is_positive_infinity(x, .xname = get_name_in_parent(x))
```

Arguments

x	Input to check.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

is_finite wraps is.finite, showing the names of the inputs in the answer. is_infinite works likewise for is.infinite. The assert_* functions return nothing but throw an error if the corresponding is_* function returns FALSE.

See Also

is.finite

Examples

```
x <- c(0, Inf, -Inf, NA, NaN)
is_finite(x)
is_infinite(x)
is_positive_infinity(x)
is_negative_infinity(x)
assert_all_are_finite(1:10)
assert_any_are_finite(c(1, Inf))
assert_all_are_infinite(c(Inf, -Inf))
assertive.base::dont_stop(assert_all_are_finite(c(0, Inf, -Inf, NA, NaN)))
```

---

assert_all_are_imaginary

Is the input real/imaginary?

Description

Checks to see if the input is real or imaginary.

Usage

```
assert_all_are_imaginary(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_imaginary(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_real(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_real(x, tol = 100 * .Machine$double.eps, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_imaginary(x, tol = 100 * .Machine$double.eps,
  .xname = get_name_in_parent(x))

is_real(x, tol = 100 * .Machine$double.eps, .xname = get_name_in_parent(x))
```

Arguments

`x` Input to check.

`tol` Imaginary/real components smaller than `tol` are not considered.

`na_ignore` A logical value. If `FALSE`, `NA` values cause an error; otherwise they do not. Like `na.rm` in many stats package functions, except that the position of the failing values does not change.

`severity` How severe should the consequences of the assertion be? Either `"stop"`, `"warning"`, `"message"`, or `"none"`.

`.xname` Not intended to be used directly.

Value

`TRUE` if the input has imaginary component equal to zero. The `assert_*` functions return nothing but throw an error if the corresponding `is_*` function returns `FALSE`.

See Also

`complex`

Examples

```
(x <- with(expand.grid(re = -1:1, im = -1:1), re + im * 1i))
is_real(x)
is_imaginary(x)

# By default, very small imaginary/real components are ignored.
x <- .Machine$double.eps * (1 + 1i)
is_real(x)
is_real(x, 0)
is_imaginary(x)
is_imaginary(x, 0)
# numbers with both a real and imaginary component return FALSE
# (since they are neither purely real nor purely imaginary)
cmplx <- 1 + 1i
is_real(cmplx)
is_imaginary(cmplx)
assert_all_are_real(1:10)
assert_all_are_real(1:10 + 0i)
assert_any_are_real(c(1i, 0))
assert_all_are_imaginary(1:10 * 1i)
assert_any_are_imaginary(c(1i, 0))
assertive.base::dont_stop(assert_all_are_real(x))
assertive.base::dont_stop(assert_all_are_imaginary(x))
```

assert_all_are_in_closed_range *Is the input in range?*

Description

Checks to see if the input is within an numeric interval.

Usage

```
assert_all_are_in_closed_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_in_closed_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_in_left_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_in_left_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_in_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_in_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_in_range(x, lower = -Inf, upper = Inf,
  lower_is_strict = FALSE, upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_in_range(x, lower = -Inf, upper = Inf,
  lower_is_strict = FALSE, upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_in_right_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_in_right_open_range(x, lower = -Inf, upper = Inf,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_negative(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_negative(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))
```

```
assert_all_are_non_negative(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_non_negative(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_non_positive(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_non_positive(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_percentages(x, lower_is_strict = FALSE,
  upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_percentages(x, lower_is_strict = FALSE,
  upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_positive(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_positive(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_proportions(x, lower_is_strict = FALSE,
  upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_proportions(x, lower_is_strict = FALSE,
  upper_is_strict = FALSE, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_in_closed_range(x, lower = -Inf, upper = Inf,
  .xname = get_name_in_parent(x))

is_in_left_open_range(x, lower = -Inf, upper = Inf,
  .xname = get_name_in_parent(x))

is_in_open_range(x, lower = -Inf, upper = Inf,
  .xname = get_name_in_parent(x))

is_in_range(x, lower = -Inf, upper = Inf, lower_is_strict = FALSE,
  upper_is_strict = FALSE, .xname = get_name_in_parent(x))

is_in_right_open_range(x, lower = -Inf, upper = Inf,
  .xname = get_name_in_parent(x))

is_negative(x, .xname = get_name_in_parent(x))

is_non_negative(x, .xname = get_name_in_parent(x))

is_non_positive(x, .xname = get_name_in_parent(x))

is_percentage(x, lower_is_strict = FALSE, upper_is_strict = FALSE,
  .xname = get_name_in_parent(x))

is_positive(x, .xname = get_name_in_parent(x))

is_proportion(x, lower_is_strict = FALSE, upper_is_strict = FALSE,
  .xname = get_name_in_parent(x))
```

Arguments

x Input to check.

lower Lower bound for the interval.

upper Upper bound for the interval.

na_ignore A logical value. If FALSE, NA values cause an error; otherwise they do not. Like na.rm in many stats package functions, except that the position of the failing values does not change.

severity How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

lower_is_strict If TRUE, the lower bound is open (strict) otherwise it is closed.

upper_is_strict If TRUE, the upper bound is open (strict) otherwise it is closed.

.xname Not intended to be used directly.

Value

The is_* functions return TRUE if the input is within an interval. The assert_* functions return nothing but throw an error if the corresponding is_* function returns FALSE.

Note

is_in_range provides the most flexibility in determining if values are within a numeric interval. The other functions restrict the input arguments for convience in common cases. For example, is_percentage forces the interval to be from 0 to 100. The function is not vectorized by the lower_is_strict and upper_is_strict for speed (these are assumed to be scalar logical values).

Examples

```
assert_all_are_positive(1:10)
assert_all_are_non_negative(0:10)
assert_any_are_positive(c(-1, 1))
assert_all_are_percentages(c(0, 50, 100))
assert_all_are_proportions(c(0, 0.5, 1))
assert_all_are_in_left_open_range(1 + .Machine$double.eps, lower = 1)
```

---

assert_all_are_nan *Is the input (not) NaN?*

Description

Checks to see if the input is a number that is(n't) NaN.

Usage

```
assert_all_are_nan(x, severity = getOption("assertive.severity", "stop"))

assert_any_are_nan(x, severity = getOption("assertive.severity", "stop"))

assert_all_are_not_nan(x, severity = getOption("assertive.severity", "stop"))

assert_any_are_not_nan(x, severity = getOption("assertive.severity", "stop"))

is_nan(x, .xname = get_name_in_parent(x))

is_not_nan(x, .xname = get_name_in_parent(x))
```

Arguments

x	Input to check.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

is_nan wraps is.nan, coercing the input to numeric if necessary. is_not_nan works similarly, but returns the negation. The assert_* functions return nothing but throw an error if the corresponding is_* function returns FALSE.

See Also

is.nan

Examples

```
x <- c(0, NaN, NA)
is_nan(x)
is_not_nan(x)
assert_all_are_not_nan(1:10)
assert_any_are_not_nan(x)
assertive.base::dont_stop(assert_all_are_not_nan(x))
```

---

assert_all_numbers_are_whole_numbers

Is the input a whole number?

Description

Checks that the (probably floating point) input is a whole number.

Usage

```
assert_all_numbers_are_whole_numbers(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_numbers_are_whole_numbers(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_all_are_whole_numbers(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_whole_numbers(x, tol = 100 * .Machine$double.eps,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

is_whole_number(x, tol = 100 * .Machine$double.eps,
  .xname = get_name_in_parent(x))
```

Arguments

x	Input to check.
tol	Differences smaller than tol are not considered.
na_ignore	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like na.rm in many stats package functions, except that the position of the failing values does not change.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

TRUE if the input is a whole number.

Note

The term whole number is used to distinguish from integer in that the input x need not have type integer. In fact it is expected that x will be numeric.

See Also

is_divisible_by

Examples

```
# 1, plus or minus a very small number
x <- 1 + c(0, .Machine$double.eps, -.Machine$double.neg.eps)
# By default, you get a bit of tolerance for rounding errors
is_whole_number(x)
# Set the tolerance to zero for exact matching.
is_whole_number(x, tol = 0)
```

Index