

# Package ‘assertive.properties’

April 21, 2022

**Type** Package

**Title** Assertions to Check Properties of Variables

**Version** 0.0-5

**Date** 2022-04-18

**Author** Richard Cotton [aut, cre]

**Maintainer** Richard Cotton <richierocks@gmail.com>

**Description** A set of predicates and assertions for checking the properties of variables, such as length, names and attributes. 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.properties>

**BugReports** <https://bitbucket.org/richierocks/assertive.properties/issues>

**Depends** R (>= 3.0.0)

**Imports** assertive.base (>= 0.0-7), methods

**Suggests** testthat

**License** GPL (>= 3)

**LazyLoad** 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** 'are-same-size.R' 'assert-are-same-size.R' 'imports.R'  
'assert-has-attributes.R' 'assert-has-dims.R'  
'assert-has-dupes.R' 'assert-has-names.R' 'assert-has-slot.R'  
'assert-is-atomic-recursive-vector.R'  
'assert-is-empty-scalar.R' 'assert-is-monotonic.R'  
'assert-is-null.R' 'assert-is-unsorted.R' 'has-attributes.R'  
'has-dims.R' 'has-dupes.R' 'has-names.R' 'has-slot.R'  
'is-atomic-recursive-vector.R' 'is-empty-scalar.R'  
'is-monotonic.R' 'is-null.R' 'is-unsorted.R' 'utils.R'

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-04-21 12:40:08 UTC

## R topics documented:

are_same_length . . . . .	2
assert_has_all_attributes . . . . .	4
assert_has_colnames . . . . .	5
assert_has_cols . . . . .	6
assert_has_dims . . . . .	7
assert_has_duplicates . . . . .	8
assert_has_elements . . . . .	9
assert_has_slot . . . . .	11
assert_is_atomic . . . . .	12
assert_is_monotonic_increasing . . . . .	14
assert_is_not_null . . . . .	15
assert_is_unsorted . . . . .	17
DIM . . . . .	18
has_any_attributes . . . . .	19
n_elements . . . . .	20
<b>Index</b>	<b>21</b>

---

are_same_length	<i>Are the inputs the same length/dimension?</i>
-----------------	--

---

### Description

Checks if the inputs are the same length, or have the same dimensions.

### Usage

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

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

```
)  
  
are_same_length_legacy(..., l = list())  
  
assert_are_same_length(  
  x,  
  y,  
  severity = getOption("assertive.severity", "stop")  
)  
  
assert_have_same_dims(x, y, severity = getOption("assertive.severity", "stop"))  
  
assert_all_are_same_length_legacy(..., l = list())  
  
assert_all_are_same_length(..., l = list())  
  
assert_any_are_same_length_legacy(..., l = list())  
  
assert_any_are_same_length(..., l = list())
```

### Arguments

x	An R object or expression.
y	Another R object or expression.
.xname	Not intended to be used directly.
.yname	Not intended to be used directly.
...	Some R expressions.
l	A list of R expressions.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

### Value

are\_same\_length and have\_same\_dims return TRUE if x and y have the same length, or their dimensions are identical. The assert\_\* functions throw an error on failure.

The legacy function are\_same\_length\_legacy allows an arbitrary number of inputs and returns a symmetric square logical matrix which is TRUE where pairs of inputs are the same length. (The new version of the function is easier to work with, and it is recommended that you switch your code to it.)

### See Also

[length](#), [are\\_identical](#)

**Examples**

```

are_same_length(runif(5), list(1, 2:3, 4:6, 7:10, 11:15))
assertive.base::dont_stop(
  assert_are_same_length(runif(6), list(1, 2:3, 4:6, 7:10, 11:15))
)
have_same_dims(
  matrix(1:12, nrow = 4),
  data.frame(x = 1:4, y = 5:8, y = 9:12)
)
have_same_dims(1:5, matrix(1:5))

```

---

```
assert_has_all_attributes
```

*Does the input have the specified attributes?*

---

**Description**

Checks to see if the input has the specified attributes.

**Usage**

```

assert_has_all_attributes(
  x,
  attrs,
  severity = getOption("assertive.severity", "stop")
)

assert_has_any_attributes(
  x,
  attrs,
  severity = getOption("assertive.severity", "stop")
)

has_attributes(x, attrs, .xname = get_name_in_parent(x))

```

**Arguments**

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

**Value**

has\_attributes returns TRUE where x has the attributes specified in attrs. assert\_has\_terms returns nothing but throws an error if has\_terms is not TRUE.

**See Also**

[has\\_any\\_attributes](#) for checking that an object has any attributes at all.

**Examples**

```
# has_attributes is vectorized on attrs
has_attributes(sleep, c("class", "names", "row.names", "col.names"))

# You can check for any or all of these attributes to be present.
x <- structure(c(a = 1), b = 2)
assert_has_all_attributes(x, c("names", "b"))
assert_has_any_attributes(x, c("names", "not an attribute"))

# These examples should fail.
assertive.base::dont_stop({
  assert_has_all_attributes(x, c("names", "not an attribute"))
})
```

---

assert\_has\_colnames *Does the input have names?*

---

**Description**

Checks to see if the input has (row/column/dimension) names.

**Usage**

```
assert_has_colnames(x, severity = getOption("assertive.severity", "stop"))
assert_has_dimnames(x, severity = getOption("assertive.severity", "stop"))
assert_has_names(x, severity = getOption("assertive.severity", "stop"))
assert_has_rownames(x, severity = getOption("assertive.severity", "stop"))

has_colnames(x, .xname = get_name_in_parent(x))
has_dimnames(x, .xname = get_name_in_parent(x))
has_names(x, .xname = get_name_in_parent(x))
has_rownames(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**

has\_names returns TRUE if names is non-null. has\_rownames, has\_colnames and has\_dimnames work in a similar fashion, checking the corresponding attributes. assert\_has\_names returns nothing but throws an error if has\_names is not TRUE.

**Note**

Empty names (i.e., "") are not allowed in R, and are not checked here.

**See Also**

[names](#), [rownames](#), [colnames](#), [dimnames](#).

**Examples**

```
assert_has_names(c(a = 1, 2))
dfr <- data.frame(x = 1:5)
assert_has_rownames(dfr)
assert_has_colnames(dfr)
assert_has_dimnames(dfr)
```

---

assert\_has\_cols

*Does the input have rows/columns?*

---

**Description**

Checks to see if the input has rows/columns.

**Usage**

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

has_cols(x, .xname = get_name_in_parent(x))
has_rows(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**

has\_rows and has\_cols return TRUE if nrow and ncol respectively return a value that is non-null and positive. The assert\_\* functions return nothing but throw an error if the corresponding has\_\* function returns FALSE.

**See Also**

[ncol](#).

**Examples**

```
assert_has_rows(data.frame(x = 1:10))
assert_has_cols(matrix())
```

---

assert_has_dims	<i>Does the input have dimensions?</i>
-----------------	--

---

**Description**

Checks to see if the input has dimensions.

**Usage**

```
assert_has_dims(x, severity = getOption("assertive.severity", "stop"))
has_dims(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**

has\_dims returns TRUE if dim is non-null. assert\_has\_dims returns nothing but throws an error if has\_dims is not TRUE.

**See Also**

[dim](#), [is\\_of\\_dimension](#).

---

assert\_has\_duplicates *Does the input have duplicates?*

---

### Description

Checks to see if the input has duplicates.

### Usage

```
assert_has_duplicates(x, severity = getOption("assertive.severity", "stop"))
assert_has_no_duplicates(x, severity = getOption("assertive.severity", "stop"))
has_duplicates(x, .xname = get_name_in_parent(x))
has_no_duplicates(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

has\_duplicates returns TRUE if anyDuplicated is TRUE. assert\_has\_duplicates returns nothing but throws an error if has\_duplicates is not TRUE. has\_no\_duplicates is the negation of has\_duplicates.

### See Also

[anyDuplicated](#).

### Examples

```
x <- sample(10, 100, replace = TRUE)
assert_has_duplicates(x)
has_no_duplicates(x)
```



---

assert\_has\_elements    *Is the input empty/scalar?*

---

**Description**

Checks to see if the input has length zero/one.

**Usage**

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

```
assert_is_empty(  
  x,  
  metric = c("length", "elements"),  
  severity = getOption("assertive.severity", "stop")  
)
```

```
assert_is_non_empty(  
  x,  
  metric = c("length", "elements"),  
  severity = getOption("assertive.severity", "stop")  
)
```

```
assert_is_non_scalar(  
  x,  
  metric = c("length", "elements"),  
  severity = getOption("assertive.severity", "stop")  
)
```

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

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

```
assert_is_scalar(  
  x,  
  metric = c("length", "elements"),  
  severity = getOption("assertive.severity", "stop")  
)
```

```
is_empty(x, metric = c("length", "elements"), .xname = get_name_in_parent(x))
```

```
is_non_empty(  
  x,
```

```

    metric = c("length", "elements"),
    .xname = get_name_in_parent(x)
)

is_non_scalar(
  x,
  metric = c("length", "elements"),
  .xname = get_name_in_parent(x)
)

is_scalar(x, metric = c("length", "elements"), .xname = get_name_in_parent(x))

has_elements(x, n, .xname = get_name_in_parent(x))

is_of_dimension(x, n, .xname = get_name_in_parent(x))

is_of_length(x, n, .xname = get_name_in_parent(x))

```

### Arguments

x	Input to check.
n	Non-negative integer(s) of the expected length/number of elements/ lengths of dimensions. See note.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
metric	A string. Should be length or the number of elements be used to determine if the object is empty/non-empty/scalar?
.xname	Not intended to be used directly.

### Value

is\_empty returns TRUE if the input has length zero. is\_scalar returns TRUE if the input has length one. The assert\_\* functions return nothing but throw an error if the corresponding is\_\* function returns FALSE.

### Note

For is\_empty, is\_non\_empty and is\_scalar, n should be an single integer representing either the expected length or the expected number of elements in x. For is\_of\_dimension n should be a vector of integers representing the expected lengths of dimensions.

### See Also

[length](#).

**Examples**

```

# is_of_length returns TRUE if the length of an object
# matches a specified value.
is_of_length(1:5, 5)
assert_is_of_length(1:5, 5)

# has_elements returns TRUE if an object has a specified
# number of elements. This is usually the same thing.
has_elements(1:5, 5)
assert_has_elements(1:5, 5)

# Data frames and lists behave differently for length
# and number of elements.
d <- data.frame(x = 1:5, y = letters[1:5])
assert_is_of_length(d, 2)
assert_has_elements(d, 10)

l <- list(a = 1:5, b = list(b.a = 1:3, b.b = 1:7))
assert_is_of_length(l, 2)
assert_has_elements(l, 15)

# Functions always have length one, but may have lots of
# elements.
assert_is_of_length(var, 1)
assert_has_elements(var, 54)

# is_scalar is a shortcut for length one, or one elements.
assert_is_scalar(99)
assert_is_scalar("Multiple words in a single string are scalar.")
assert_is_scalar(NA)

# The two metrics can yield different results!
is_scalar(list(1:5))
is_scalar(list(1:5), "elements")
is_scalar(var)
is_scalar(var, "elements")

# Similarly, is_empty is a shortcut for length zero/zero elements.
assert_is_empty(NULL)
assert_is_empty(numeric())
assert_is_non_empty(1:10)
assert_is_non_empty(NA)

# is_of_dimension tests the lengths of all dimensions.
assert_is_of_dimension(d, c(5, 2))
assert_is_of_dimension(l, NULL)

```

**Description**

Checks to see if the object is an S4 object with a particular slot.

**Usage**

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

```
has_slot(x, slotname, .xname = get_name_in_parent(x))
```

**Arguments**

x	Input to check. Intended to be an S4 object.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
slotname	A string naming a slot to check for.
.xname	Not intended to be used directly.

**Value**

has\_names returns TRUE if names is non-null.

**See Also**

[slot](#)

**Examples**

```
setClass("numbers", representation(foo = "numeric"))
x <- new("numbers", foo = 1:10)
has_slot(x, "foo")
has_slot(x, "bar")
has_slot(1:10, "foo")
```

---

assert\_is\_atomic

*Is the input atomic/recursive/vector?*

---

**Description**

Checks to see if the input is a type that is atomic/recursive/vector.

**Usage**

```
assert_is_atomic(x, severity = getOption("assertive.severity", "stop"))  
assert_is_nested(x, severity = getOption("assertive.severity", "stop"))  
assert_is_non_nested(x, severity = getOption("assertive.severity", "stop"))  
assert_is_recursive(x, severity = getOption("assertive.severity", "stop"))  
assert_is_vector(x, severity = getOption("assertive.severity", "stop"))  
  
is_atomic(x, .xname = get_name_in_parent(x))  
is_nested(x, .xname = get_name_in_parent(x))  
is_non_nested(x, .xname = get_name_in_parent(x))  
is_recursive(x, .xname = get_name_in_parent(x))  
is_vector(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\_atomic, is\_recursive and is\_vector wrap is.atomic, is.recursive and is.vector respectively, providing more information on failure. is\_nested checks for recursive objects where at least one element is also recursive. is\_non\_nested returns TRUE for atomic objects and recursive objects where no elements are recursive. The assert\_\* functions return nothing but throw an error if the corresponding is\_\* function returns FALSE.

**See Also**

[is.atomic](#) and [is.vector](#).

**Examples**

```
atomic_types <- list(  
  logical(),  
  integer(),  
  numeric(),  
  complex(),  
  character(),  
  raw(),
```

```

    matrix(),
    array(),
    factor(),
    NULL
  )
  for(var in atomic_types) assert_is_atomic(var)

  recursive_types <- list(
    list(),
    expression(),
    data.frame(),
    y ~ x,
    function(){},
    call("sin", "pi")
  )
  for(var in recursive_types) assert_is_recursive(var)

  # Names are neither atomic nor recursive
  a_name <- as.name("x")
  is_atomic(a_name)
  is_recursive(a_name)

  vector_types <- c(
    atomic_types[1:6],
    recursive_types[1:2]
  )
  for(var in vector_types) assert_is_vector(var)

  # Nested objects are recursive and have at least one recursive element
  nested_list <- list(a = 1, b = list(2:3))
  assert_is_nested(nested_list)
  for(elt in nested_list) assert_is_non_nested(elt)

```

---

```
assert_is_monotonic_increasing
```

*Is the vector monotonically increasing or decreasing?*

---

### Description

Checks to see if the input is monotonically increasing or decreasing.

### Usage

```

assert_is_monotonic_increasing(
  x,
  strictly = FALSE,
  severity = getOption("assertive.severity", "stop")
)

```

```

assert_is_monotonic_decreasing(
  x,
  strictly = FALSE,
  severity = getOption("assertive.severity", "stop")
)

is_monotonic_increasing(x, strictly = FALSE, .xname = get_name_in_parent(x))

is_monotonic_decreasing(x, strictly = FALSE, .xname = get_name_in_parent(x))

```

### Arguments

x	Input to check.
strictly	Logical. If TRUE, the input is checked for being strictly monotonic; that is, consecutive values cannot be equal.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

### Examples

```

x <- c(1, 2, 2, 1, 3, 2)
is_monotonic_increasing(x)
is_monotonic_increasing(x, TRUE)
is_monotonic_decreasing(x)
is_monotonic_decreasing(x, TRUE)

# Also works with, e.g., dates & times
is_monotonic_increasing(Sys.time() + x)

# These checks should fail
assertive.base::dont_stop({
  assert_is_monotonic_increasing(x)
  assert_is_monotonic_decreasing(x)
})

```

---

assert\_is\_not\_null     *Is the input (not) null?*

---

### Description

Checks to see if the input is (not) null.

**Usage**

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

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

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

```
is_null(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_null` wraps `is.null`, providing more information on failure. `is_not_null` returns TRUE in the opposite case. The `assert_*` functions return nothing but throw an error if the corresponding `is_*` function returns FALSE.

**See Also**

[is.null](#).

**Examples**

```
# Predicate for NULL.
is_null(NULL)
is_null(c())

# Atomic vectors of length zero are not NULL!
is_null(numeric())
# ... and neither is NA
is_null(NA)

# The opposite check
is_not_null(NULL)
is_not_null(c())
is_not_null(numeric())

# These checks should pass
assert_is_null(NULL)
assert_is_null(c())
assert_is_not_null(NA)

# This should fail
assertive.base::dont_stop(assert_is_null(NaN))
```



---

assert_is_unsorted	<i>Is the input unsorted?</i>
--------------------	-------------------------------

---

### Description

Checks to see if the input is unsorted (without the cost of sorting it).

### Usage

```
assert_is_unsorted(  
  x,  
  na.rm = FALSE,  
  strictly = FALSE,  
  severity = getOption("assertive.severity", "stop")  
)  
  
is_unsorted(x, na.rm = FALSE, strictly = FALSE, .xname = get_name_in_parent(x))
```

### Arguments

x	Input to check.
na.rm	If TRUE, remove NAs before checking.
strictly	If TRUE, equal values count as unsorted.
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\_unsorted reimplements is.unsorted, providing more information on failure. assert\_is\_unsorted returns nothing but throws an error if is\_unsorted returns FALSE.

### Note

The builtin function is.unsorted usually returns NA when the input is recursive and has length 2, though for some classes (particularly data.frames) it returns a TRUE or FALSE value. The logic behind those is difficult to interpret, and gives odd results, so is\_unsorted always returns NA in this case.

### See Also

[is.unsorted](#).

**Examples**

```
assert_is_unsorted(c(1, 3, 2))
assert_is_unsorted(c(1, 1, 2), strictly = TRUE)
# These checks should fail.
assertive.base::dont_stop({
  assert_is_unsorted(c(1, 1, 2))
  assert_is_unsorted(c(2, 1, 0))
})
```

---

**DIM***Get the dimensions of an object*

---

**Description**

Get the dimensions of an object, returning the length if that object has no dim attribute.

**Usage**

```
DIM(x)
```

**Arguments**

x                   Any object.

**Value**

A integer vector of non-negative values.

**See Also**

[NROW](#), [dim](#)

**Examples**

```
# For data frames and matrices, DIM is the same as dim.
DIM(sleep)
# For vectors (and other objects without a dim attribute), DIM is the
# same as length.
DIM(1:10)
DIM(list(x = 1:10))
```

---

has_any_attributes	<i>Does the input have any attributes?</i>
--------------------	--

---

### Description

Checks to see if the input has any attributes.

### Usage

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

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

### Arguments

x	Input to check.
.xname	Not intended to be used directly.

### Value

has\_any\_attributes returns TRUE if attributes(x) has length greater than zero. has\_attributes returns a logical vector that is TRUE whenever the specified attribute is not NULL.

### Note

There are no corresponding assert functions, since they overlap too closely with the assertions for [has\\_attributes](#).

### See Also

[has\\_attributes](#) to check for specific attributes.

### Examples

```
has_any_attributes(matrix())  
has_no_attributes(data.frame())
```

---

n_elements	<i>Get the number of elements</i>
------------	-----------------------------------

---

**Description**

Gets the number of elements in an object.

**Usage**

```
n_elements(x)
```

**Arguments**

x                    Any object.

**Value**

A non-negative integer of the number of elements.

**Note**

For atomic objects, the number of elements is the product of the dimensions, as calculated by [DIM](#). For recursive objects, the number of elements is the sum of the number of elements of each of their atomic components.

**See Also**

[DIM](#)

**Examples**

```
n_elements(1:10)
n_elements(NULL)
n_elements(data.frame(x = 1:5, y = rnorm(5)))
n_elements(list(1:5, list(1:3, list(1:7))))
n_elements(var) # depends upon the length of the body
```

# Index

anyDuplicated, 8  
are\_identical, 3  
are\_same\_length, 2  
are\_same\_length\_legacy  
    (are\_same\_length), 2  
assert\_all\_are\_same\_length  
    (are\_same\_length), 2  
assert\_all\_are\_same\_length\_legacy  
    (are\_same\_length), 2  
assert\_any\_are\_same\_length  
    (are\_same\_length), 2  
assert\_any\_are\_same\_length\_legacy  
    (are\_same\_length), 2  
assert\_are\_same\_length  
    (are\_same\_length), 2  
assert\_has\_all\_attributes, 4  
assert\_has\_any\_attributes  
    (assert\_has\_all\_attributes), 4  
assert\_has\_colnames, 5  
assert\_has\_cols, 6  
assert\_has\_dimnames  
    (assert\_has\_colnames), 5  
assert\_has\_dims, 7  
assert\_has\_duplicates, 8  
assert\_has\_elements, 9  
assert\_has\_names (assert\_has\_colnames),  
    5  
assert\_has\_no\_duplicates  
    (assert\_has\_duplicates), 8  
assert\_has\_rownames  
    (assert\_has\_colnames), 5  
assert\_has\_rows (assert\_has\_cols), 6  
assert\_has\_slot, 11  
assert\_have\_same\_dims  
    (are\_same\_length), 2  
assert\_is\_atomic, 12  
assert\_is\_empty (assert\_has\_elements), 9  
assert\_is\_monotonic\_decreasing  
    (assert\_is\_monotonic\_increasing),  
    14  
assert\_is\_monotonic\_increasing, 14  
assert\_is\_nested (assert\_is\_atomic), 12  
assert\_is\_non\_empty  
    (assert\_has\_elements), 9  
assert\_is\_non\_nested  
    (assert\_is\_atomic), 12  
assert\_is\_non\_scalar  
    (assert\_has\_elements), 9  
assert\_is\_not\_null, 15  
assert\_is\_null (assert\_is\_not\_null), 15  
assert\_is\_of\_dimension  
    (assert\_has\_elements), 9  
assert\_is\_of\_length  
    (assert\_has\_elements), 9  
assert\_is\_recursive (assert\_is\_atomic),  
    12  
assert\_is\_scalar (assert\_has\_elements),  
    9  
assert\_is\_unsorted, 17  
assert\_is\_vector (assert\_is\_atomic), 12  
colnames, 6  
DIM, 18, 20  
dim, 7, 18  
dimnames, 6  
has\_any\_attributes, 5, 19  
has\_attributes, 19  
has\_attributes  
    (assert\_has\_all\_attributes), 4  
has\_colnames (assert\_has\_colnames), 5  
has\_cols (assert\_has\_cols), 6  
has\_dimnames (assert\_has\_colnames), 5  
has\_dims (assert\_has\_dims), 7  
has\_duplicates (assert\_has\_duplicates),  
    8  
has\_elements (assert\_has\_elements), 9  
has\_names (assert\_has\_colnames), 5

has\_no\_attributes (has\_any\_attributes),  
19

has\_no\_duplicates  
(assert\_has\_duplicates), 8

has\_rownames (assert\_has\_colnames), 5

has\_rows (assert\_has\_cols), 6

has\_slot (assert\_has\_slot), 11

have\_same\_dims (are\_same\_length), 2

is.atomic, 13

is.null, 16

is.unsorted, 17

is.vector, 13

is\_atomic (assert\_is\_atomic), 12

is\_empty (assert\_has\_elements), 9

is\_monotonic  
(assert\_is\_monotonic\_increasing),  
14

is\_monotonic\_decreasing  
(assert\_is\_monotonic\_increasing),  
14

is\_monotonic\_increasing  
(assert\_is\_monotonic\_increasing),  
14

is\_nested (assert\_is\_atomic), 12

is\_non\_empty (assert\_has\_elements), 9

is\_non\_nested (assert\_is\_atomic), 12

is\_non\_scalar (assert\_has\_elements), 9

is\_not\_null (assert\_is\_not\_null), 15

is\_null (assert\_is\_not\_null), 15

is\_of\_dimension, 7

is\_of\_dimension (assert\_has\_elements), 9

is\_of\_length (assert\_has\_elements), 9

is\_recursive (assert\_is\_atomic), 12

is\_scalar (assert\_has\_elements), 9

is\_unsorted (assert\_is\_unsorted), 17

is\_vector (assert\_is\_atomic), 12

length, 3, 10

n\_elements, 20

names, 6

ncol, 7

NROW, 18

rownames, 6

slot, 12