

Package ‘assertive.base’

February 8, 2021

Type Package

Title A Lightweight Core of the 'assertive' Package

Version 0.0-9

Date 2021-02-07

Author Richard Cotton [aut, cre],
Sunkyu Choi [trl],
Ivanka Skakun [trl],
Gergely Dar<c3><b3>czi [trl],
Anton Antonov [trl],
Hisham Ben Hamidane [trl],
Anja Billing [trl],
Aditya Bhagwat [trl],
Rasmus B<c3><a5><c3><a5>th [trl],
Mine Cetinkaya-Rundel [trl],
Aspasia Chatziefthymiou [trl]

Maintainer Richard Cotton <richierocks@gmail.com>

Description A minimal set of predicates and assertions used by the assertive package. 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.base>

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

Depends R (>= 3.5.0)

Imports methods, utils

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 <<https://qatar-weill.cornell.edu>>. The Core is supported by

'Biomedical Research Program' funds, a program funded by Qatar Foundation. Ukrainian translations were supported by www.coupofy.com.

RoxygenNote 7.1.1

ByteCompile true

NeedsCompilation no

Repository CRAN

Date/Publication 2021-02-08 05:50:02 UTC

R topics documented:

are_identical	2
assertionError	4
assert_engine	5
bapply	6
call_and_name	7
cause	8
coerce_to	8
dont_stop	9
false	10
get_name_in_parent	11
is2	11
merge.list	12
merge_dots_with_list	13
na	14
parenthesize	15
print.scalar_with_cause	16
print_and_capture	16
safe_deparse	17
set_cause	18
strip_attributes	19
Truth	19
use_first	23
Index	24

are_identical	<i>Are the inputs identical?</i>
---------------	----------------------------------

Description

Checks if the inputs are identical.

Usage

```

are_identical(
  x,
  y,
  allow_attributes = FALSE,
  .xname = get_name_in_parent(x),
  .yname = get_name_in_parent(y)
)

are_identical_legacy(..., l = list())

assert_are_identical(
  x,
  y,
  allow_attributes = FALSE,
  severity = getOption("assertive.severity", "stop")
)

assert_all_are_identical_legacy(..., l = list())

assert_any_are_identical_legacy(..., l = list())

```

Arguments

<code>x</code>	An R object or expression.
<code>y</code>	Another R object or expression.
<code>allow_attributes</code>	If TRUE, The attributes of <code>x</code> and <code>y</code> are allowed to differ.
<code>.xname</code>	Not intended to be used directly.
<code>.yname</code>	Not intended to be used directly.
<code>...</code>	Some R expressions, deprecated.
<code>l</code>	A list of R expressions, deprecated.
<code>severity</code>	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

Value

`are_identical` returns TRUE if `x` and `y` are identical. The `assert_*` function throws an error on failure.

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

See Also

[identical](#), [are_same_length](#)

Examples

```
x <- 1:5
are_identical(c(1, -1), cos(c(0, pi)))
assertive.base::dont_stop(assert_are_identical(c(1, 1), cos(c(0, pi))))
```

assertionError	<i>Condition classes</i>
----------------	--------------------------

Description

Error, warning, and message classes derived from their simple equivalents.

Usage

```
assertionError(message, call = NULL, predicate_name = NULL)
assertionWarning(message, call = NULL, predicate_name = NULL)
assertionMessage(message, call = NULL, predicate_name = NULL)
```

Arguments

message A string describing the problem.
 call A call describing the source of the condition.
 predicate_name A string naming the predicate that was called when the condition occurred.

Value

An object of class `assertionError`, `assertionWarning`, or `assertionMessage`.

Note

These objects behave the same as the standard-issue `simpleError`, `simpleWarning`, and `simpleMessage` objects from base-R. The extra class allows you to provide custom handling for assertions inside `tryCatch`.

Examples

```
tryCatch(
  assert_all_are_true(FALSE),
  error = function(e)
  {
    if(inherits(e, "assertionCondition"))
    {
      # Handle assertions
      message("This is an assertion condition.")

      # Handle assertions cause by a specific predicate
```

```

        if(e$predicate_name == "is_true")
        {
        }
    } else
    {
        # Handle other error types
    }
}
)

```

assert_engine

Throws an error if a condition isn't met

Description

The workhorse of the package that creates an assertion from a predicate. If a condition isn't met, then an error is thrown. This function is exported for use by package developers so that they can create their own assert functions.

Usage

```

assert_engine(
  predicate,
  ...,
  msg = "The assertion failed.",
  what = c("all", "any"),
  na_ignore = FALSE,
  severity = c("stop", "warning", "message", "none")
)

```

Arguments

predicate	Function that returns a logical value (possibly a vector).
...	Passed to the predicate function.
msg	The error message, in the event of failure.
what	Either 'all' or 'any', to reduce vectorised tests to a single value.
na_ignore	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like <code>na.rm</code> 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".

Value

FALSE with the attribute message, as provided in the input.

Note

Missing values are considered as FALSE for the purposes of whether or not an error is thrown.

Examples

```
# Basic usage is like do.call; pass a predicate and the arguments to it.
dont_stop(assert_engine(is_true, c(TRUE, FALSE, NA)))

# Customise the error message
dont_stop(
  assert_engine(is_true, c(TRUE, FALSE, NA), msg = "Not everything is true")
)

# Only fail when no values match the predicate's conditions
dont_stop(assert_engine(is_true, logical(3), what = "any"))

# You can use base predicates, but the error message isn't as informative
dont_stop(assert_engine(is.matrix, 1:5))

# Reduce the severity of failure
assert_engine(is_true, c(TRUE, FALSE, NA), severity = "message")
```

bapply

Wrapper to vapply that returns booleans

Description

Wrapper to [vapply](#) for functions that return a boolean (logical scalar) value.

Usage

```
bapply(x, predicate, ...)
```

Arguments

x	A vector (atomic or list).
predicate	A predicate (function that returns a bool) to apply. elementwise to x.
...	Passed to vapply.

Value

A logical vector.

Note

USE.NAMES is set to TRUE

See Also

[vapply](#).

call_and_name	<i>Call a function, and give the result names.</i>
---------------	--

Description

Calls a function, and names the result with the first argument.

Usage

```
call_and_name(fn, x, ...)
```

Arguments

fn	A function to call. See note below.
x	The first input to fn.
...	Optional additional inputs to fn.

Value

The result of `fn(x, ...)`, with names given by the argument `x`.

Note

The function, `fn`, should return an object with the same length as the input `x`. For speed and simplicity, this isn't checked; it is up to the developer of the assertion to make sure that this condition holds.

See Also

[cause](#) and [na](#).

Examples

```
call_and_name(is.finite, c(1, Inf, NA))

# Make sure that the output is the same size as the input.
# Don't do this:
dont_stop(call_and_name(isTRUE, list(TRUE, FALSE, NA)))
# Do this instead:
call_and_name(
  Vectorize(isTRUE, SIMPLIFY = FALSE),
  list(TRUE, FALSE, NA)
)
```

cause	<i>Get or set the "cause" attribute</i>
-------	---

Description

Gets or sets the "cause" (of failure) attribute of a variable.

Usage

```
cause(x)
```

```
cause(x) <- value
```

Arguments

x Any variable.

value Passed to `gettextf` and stored in the "cause" attribute.

Value

The get method returns the "cause" attribute.

See Also

[set_cause](#)

Examples

```
# Scalar case
yn <- is_identical_to_true(FALSE)
cause(yn)

# Vector case
yn <- is_true(c(TRUE, FALSE, NA))
cause(yn)
```

coerce_to	<i>Coerce variable to a different class</i>
-----------	---

Description

Coerce the input to a different class, with a warning. More reliable than [as](#), and supports coercion to multiple classes.

Usage

```
coerce_to(x, target_class, .xname = get_name_in_parent(x))
```


Arguments

x	Input to coerce.
target_class	The desired class of x. Multiple values allowed (see note).
.xname	Not intended to be used directly.

Value

The input x after attempted coercion to the target class.

Note

If x does not already have the target class, a warning is given before coercion. The function will try and convert the x to each of the classes given in target_class, in order, until it succeeds or runs out of classes to try. It will first try and convert x using a dedicated as.target_class function if that exists. If it does not exist, or throws an error then coerce_to will try to use as(x, target_class).

See Also

[is](#) and [as](#).

Examples

```
# Numbers can be coerced to characters but not to calls.
dont_stop(coerce_to(1:5, c("call", "character")))
```

dont_stop

Run code without stopping

Description

Runs code without stopping for warnings or errors.

Usage

```
dont_stop(expr)
```

Arguments

expr	Code to execute.
------	------------------

Value

A list containing the results of evaluating each call in expr.

Note

This function is dangerous, since it overrides warnings and errors. Its intended use is for documenting examples of warnings and errors.

See Also

[warning](#) and [stop](#) for generating warnings and errors respectively; [try](#) and [conditions](#) for handling them.

Examples

```
dont_stop({  
  warning("a warning")  
  x <- 1  
  stop("an error")  
  y <- sqrt(exp(x + 1))  
  assert_is_identical_to_true(y)  
  y > 0  
})
```

false

FALSE, with a cause of failure.

Description

Always returns the value FALSE, with a cause attribute.

Usage

```
false(...)
```

Arguments

... Passed to `gettextf` to create a cause of failure message.

Value

FALSE with the attribute `cause`, as provided in the input.

See Also

[cause](#) and [na](#).

get_name_in_parent *Get the name of a variable in the parent frame*

Description

Gets the name of the input in the parent frame.

Usage

```
get_name_in_parent(x, escape_percent = TRUE)
```

Arguments

`x` Variable to get the name of.

`escape_percent` Logical. If TRUE, percent signs are doubled, making the value suitable for use with `sprintf` (and hence by `false` and `na`).

Value

A string giving the name of the input in the parent frame.

Examples

```
outside <- 1
f <- function(inside, escape_percent)
{
  get_name_in_parent(inside, escape_percent)
}
f(outside, TRUE)
f('10%', TRUE)
f('10%', FALSE)
```

is2 *Alternative version of is*

Description

If a function named `is.class` exists, call `is.class(x)`. If not, call `is(x,class)`.

Usage

```
is2(x, class, .xname = get_name_in_parent(x))
```

Arguments

x	Input to check.
class	Target class that x maybe belong to.
.xname	Not intended to be used directly.

Value

TRUE if x belongs to the class and FALSE otherwise.

See Also

[is](#), and [assert_is_all_of](#) for the corresponding assert fns.

Examples

```
is2(1:5, "character")
is2(matrix(1:5), "character")
is2(1:5, c("character", "list", "numeric"))
is2(mean, c("function", "data.frame"))
```

merge.list

Merge two lists

Description

Merges two lists, taking duplicated elements from the first list.

Usage

```
## S3 method for class 'list'
merge(x, y, warn_on_dupes = TRUE, allow_unnamed_elements = FALSE, ...)
```

Arguments

x	A list.
y	A list.
warn_on_dupes	TRUE or FALSE. Should a warning be given if both x and y have elements with the same name. See note.
allow_unnamed_elements	TRUE or FALSE. Should unnamed elements be allowed?
...	Ignored.

Value

A list, combining elements from x and y.

Note

In the event of elements that are duplicated between x and y, the versions from x are used.

See Also

[merge_dots_with_list](#), [merge](#)

Examples

```
merge(  
  list(foo = 1, bar = 2, baz = 3),  
  list(foo = 4, baz = 5, quux = 6)  
)  
  
# If unnamed elements are allowed, they are included at the end  
merge(  
  list("a", foo = 1, "b", bar = 2, baz = 3, "c"),  
  list(foo = 4, "a", baz = 5, "b", quux = 6, "d"),  
  allow_unnamed_elements = TRUE  
)
```

`merge_dots_with_list` *Merge ellipsis args with a list.*

Description

Merges variable length ellipsis arguments to a function with a list argument.

Usage

```
merge_dots_with_list(  
  ...,  
  l = list(),  
  warn_on_dupes = TRUE,  
  allow_unnamed_elements = FALSE  
)
```

Arguments

<code>...</code>	Some inputs.
<code>l</code>	A list.
<code>warn_on_dupes</code>	TRUE or FALSE. Should a warning be given if both x and y have elements with the same name. See note.
<code>allow_unnamed_elements</code>	TRUE or FALSE. Should unnamed elements be allowed?

Value

A list containing the merged inputs.

Note

If any arguments are present in both the `...` and `l` arguments, the `...` version takes preference, and a warning is thrown.

See Also

[merge.list](#), [merge](#)

Examples

```
merge_dots_with_list(  
  foo = 1,  
  bar = 2,  
  baz = 3,  
  l = list(foo = 4, baz = 5, quux = 6)  
)
```

na

NA, with a cause of failure.

Description

Always returns the value (logical) NA, with a cause attribute.

Usage

```
na(...)
```

Arguments

`...` Passed to `gettextf` to create a cause of failure message.

Value

NA with the attribute `cause`, as provided in the input.

See Also

[cause](#) and [false](#).

parenthesize	<i>Wrap a string in brackets</i>
--------------	----------------------------------

Description

Parenthesise a character vector by wrapping elements in brackets, dashes or commas.

Usage

```
parenthesize(  
  x,  
  type = c("round_brackets", "square_brackets", "curly_brackets", "angle_brackets",  
           "chevrons", "hyphens", "en_dashes", "em_dashes", "commas")  
)  
  
parenthesise(  
  x,  
  type = c("round_brackets", "square_brackets", "curly_brackets", "angle_brackets",  
           "chevrons", "hyphens", "en_dashes", "em_dashes", "commas")  
)
```

Arguments

x	Character vector to wrap in parentheses.
type	String naming the type of parenthesis.

Value

A character vector of the input wrapped in parentheses.

Note

English grammar terminology is awfully confusing. The verb 'to parenthesise' means to wrap a phrase in brackets or dashes or commas, thus denoting it as supplementary material that could be left out. A 'parenthesis' as a noun is often used as a synonym for a round bracket.

See Also

[sQuote](#)

Examples

```
paste("There were three", parenthesise(3), "mice in the experiment.")  
paste(  
  "I love parmos",  
  parenthesise("Teesside's finest culinary invention", "en_dashes"),  
  "but they are sure to give me heart disease."  
)
```

```

parenthesise(letters[1:5], "curly")
paste0(
  "The R language",
  parenthesise("an offshoot of S and Scheme", "commas"),
  "is quite good for data analysis."
)

```

```
print.scalar_with_cause
```

Print methods for objects with a cause attribute

Description

Prints objects of class `scalar_with_cause` and `vector_with_cause`.

Usage

```

## S3 method for class 'scalar_with_cause'
print(x, ...)

## S3 method for class 'vector_with_cause'
print(x, na_ignore = FALSE, n_to_show = 10, ...)

```

Arguments

<code>x</code>	an object of class <code>scalar_with_cause</code> or <code>vector_with_cause</code> .
<code>...</code>	Currently unused.
<code>na_ignore</code>	A logical value. If <code>FALSE</code> , NA values are printed; otherwise they do not. Like <code>na.rm</code> in many stats package functions, except that the position of the failing values does not change.
<code>n_to_show</code>	A natural number. The maximum number of failures to show.

```
print_and_capture
```

Print a variable and capture the output

Description

Prints a variable and captures the output, collapsing the value to a single string.

Usage

```
print_and_capture(x, ...)
```


Arguments

x A variable.
... Arguments passed to [print](#) methods.

Value

A string.

See Also

[print](#), [capture.output](#)

Examples

```
# This is useful for including data frames in warnings or errors  
message("This is the sleep dataset:\n", print_and_capture(sleep))
```

safe_deparse	<i>Safe version of deparse</i>
--------------	--------------------------------

Description

A version of [deparse](#) that is guaranteed to always return a single string.

Usage

```
safe_deparse(expr, ...)
```

Arguments

expr Any R expression.
... Passed to [deparse](#).

Value

A character vector of length one.

Note

By default the RStudio IDE truncates output in the console at 1000 characters. Consequently, if you use `safe_deparse` on large or complex objects, you won't see the full value. You can change the setting using Tools -> "Global Options..." -> Code -> Display -> Console -> "Limit length of lines displayed in console to:".

Examples

```
# safe_deparse only differs from deparse when the deparse string is longer
# than width.cutoff
deparse(CO2, width.cutoff = 500L) # has length 6
safe_deparse(CO2)                 # has length 1
```

set_cause	<i>Set a cause and return the input</i>
-----------	---

Description

Sets the cause attribute of an object and returns that object.

Usage

```
set_cause(x, false_value, missing_value = "missing")
```

Arguments

`x` A variable.

`false_value` A character vector to set the cause to, where `x` is FALSE.

`missing_value` A character vector to set the cause to, where `x` is NA.

Details

If `x` is TRUE everywhere, this returns the input without setting a cause. Otherwise, the cause is an empty string where `x` is TRUE, `false_value` where it is FALSE, and `missing_value` where it is NA.

Value

`x`, with a new cause attribute.

See Also

[cause](#), [setNames](#)

strip_attributes	<i>Strip all attributes from a variable</i>
------------------	---

Description

Strips all the attributes from a variable.

Usage

```
strip_attributes(x)
```

Arguments

x Input to strip.

Value

x, without attributes.

Examples

```
x <- structure(c(foo = 1, bar = 2), some_attr = 3)
x2 <- strip_attributes(x)
attributes(x)
attributes(x2)
```

Truth	<i>Is the input TRUE/FALSE/NA?</i>
-------	------------------------------------

Description

Checks to see if the input is TRUE, FALSE or NA.

Usage

```
assert_is_identical_to_false(  
  x,  
  allow_attributes = FALSE,  
  severity = getOption("assertive.severity", "stop")  
)
```

```
assert_is_identical_to_na(  
  x,  
  allow_attributes = FALSE,  
  severity = getOption("assertive.severity", "stop")  
)
```

```
assert_is_identical_to_true(  
  x,  
  allow_attributes = FALSE,  
  severity = getOption("assertive.severity", "stop")  
)  
  
assert_all_are_false(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_false(x, severity = getOption("assertive.severity", "stop"))  
assert_all_are_na(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_na(x, severity = getOption("assertive.severity", "stop"))  
assert_all_are_true(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_true(x, severity = getOption("assertive.severity", "stop"))  
assert_all_are_not_false(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_not_false(x, severity = getOption("assertive.severity", "stop"))  
assert_all_are_not_na(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_not_na(x, severity = getOption("assertive.severity", "stop"))  
assert_all_are_not_true(x, severity = getOption("assertive.severity", "stop"))  
assert_any_are_not_true(x, severity = getOption("assertive.severity", "stop"))  
  
is_identical_to_false(  
  x,  
  allow_attributes = FALSE,  
  .xname = get_name_in_parent(x)  
)  
  
is_identical_to_na(x, allow_attributes = FALSE, .xname = get_name_in_parent(x))  
  
is_identical_to_true(  
  x,  
  allow_attributes = FALSE,  
  .xname = get_name_in_parent(x)  
)  
  
is_false(x, .xname = get_name_in_parent(x))  
  
is_na(x, coerce_to_logical = FALSE, .xname = get_name_in_parent(x))
```

```
is_not_na(x, coerce_to_logical = FALSE, .xname = get_name_in_parent(x))

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

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

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

Arguments

<code>x</code>	Input to check. See note.
<code>allow_attributes</code>	If TRUE, a scalar value of TRUE with attributes is allowed.
<code>severity</code>	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
<code>.xname</code>	Not intended to be used directly.
<code>coerce_to_logical</code>	Logical: should the input be coerced to logical before checking? See note.

Value

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

Note

`is_identical_to_true` wraps the base function `isTRUE`, providing more information on failure. Likewise, `is_identical_to_false` checks that the input is identical to FALSE. If `allow_attributes` is TRUE, a scalar value of TRUE with attributes is allowed. `is_true` and `is_false` are vectorized, returning TRUE when the inputs are TRUE and FALSE respectively.

The for `is_true`, `is_false`, `is_not_true` and `is_not_false`, `x` argument will be coerced to be a logical vector if it isn't already.

Coercion to logical is optional for `is_na` and `is_not_na`. If you do coerce, it means that `is_na` differs in behaviour from `base::is.na` for character vector, list and data frame inputs. To replicate the behaviour of `is.na`, ensure the argument `coerce_to_logical` is FALSE (this is the default).

Note that in assertive version 0.1-4 and prior, `is_identical_to_true/false` were named `is_true/false` and the vectorized versions were not present.

See Also

[isTRUE](#).

Examples

```
# Checks against logical values using base::identical
assert_is_identical_to_true(TRUE)
assert_is_identical_to_false(FALSE)
```

```

assert_is_identical_to_na(NA)

# Other NA types match
assert_is_identical_to_na(NA_complex_)

# NaN is not NA
dont_stop(assert_is_identical_to_na(NaN))

# For a slightly less strict test, you can ignore attributes
assert_is_identical_to_true(c(truth = TRUE), allow_attributes = TRUE)
assert_is_identical_to_false(matrix(FALSE), allow_attributes = TRUE)
assert_is_identical_to_na(structure(NA, class = "nanana"), allow_attributes = TRUE)

# Vectorized predicates (package name explicitly given to prevent
# problems with testthat name clash)
x <- c(TRUE, FALSE, NA)
assertive.base::is_true(x)
assertive.base::is_false(x)
is_na(x)

# ...and their opposites
is_not_true(x)
is_not_false(x)
is_not_na(x)

# Check that at least one element fits the condition
assert_any_are_true(x)
assert_any_are_false(x)
assert_any_are_na(x)

# These checks should fail:
dont_stop({
  assert_is_identical_to_true(c(truth = TRUE))
  assert_is_identical_to_true(1)
  assert_is_identical_to_true(c(TRUE, TRUE))
  assert_is_identical_to_false(matrix(FALSE))
  assert_is_identical_to_na(structure(NA, class = "nanana"))
  assert_all_are_true(x)
  assert_all_are_false(x)
  assert_all_are_na(x)
})

# base::is.na has non-standard behaviour for data.frames and lists.
# is_na and is_not_na coerce to logical vectors (except character input).
# unlist the input or use an apply function.
d <- data.frame(
  x = c(TRUE, FALSE, NA),
  y = c(0, NA, 2),
  z = c("a", "NA", NA)
)
is.na(d)
is_na(unlist(d))

```

use_first	<i>Only use the first element of a vector</i>
-----------	---

Description

If the input is not scalar, then only the first element is returned, with a warning.

Usage

```
use_first(x, indexer = c("[[", "["), .xname = get_name_in_parent(x))
```

Arguments

x	Input that should be scalar.
indexer	Either double indexing, "[[" (the default) or single indexing "[".
.xname	Not intended to be used directly.

Value

If x is scalar, it is returned unchanged, otherwise only the first element is returned, with a warning.

Examples

```
dont_stop(use_first(1:5))
```

Index

are_identical, 2
are_identical_legacy (are_identical), 2
are_same_length, 3
as, 8, 9
assert_all_are_false (Truth), 19
assert_all_are_identical_legacy
 (are_identical), 2
assert_all_are_na (Truth), 19
assert_all_are_not_false (Truth), 19
assert_all_are_not_na (Truth), 19
assert_all_are_not_true (Truth), 19
assert_all_are_true (Truth), 19
assert_any_are_false (Truth), 19
assert_any_are_identical_legacy
 (are_identical), 2
assert_any_are_na (Truth), 19
assert_any_are_not_false (Truth), 19
assert_any_are_not_na (Truth), 19
assert_any_are_not_true (Truth), 19
assert_any_are_true (Truth), 19
assert_are_identical (are_identical), 2
assert_engine, 5
assert_is_all_of, 12
assert_is_identical_to_false (Truth), 19
assert_is_identical_to_na (Truth), 19
assert_is_identical_to_true (Truth), 19
assertionError, 4
assertionMessage (assertionError), 4
assertionWarning (assertionError), 4

bapply, 6

call_and_name, 7
capture.output, 17
cause, 7, 8, 10, 14, 18
cause<- (cause), 8
coerce_to, 8
conditions, 10

deparse, 17

dont_stop, 9

false, 10, 14

get_name_in_parent, 11

identical, 3
is, 9, 12
is2, 11
is_false (Truth), 19
is_identical_to_false (Truth), 19
is_identical_to_na (Truth), 19
is_identical_to_true (Truth), 19
is_na (Truth), 19
is_not_false (Truth), 19
is_not_na (Truth), 19
is_not_true (Truth), 19
is_true (Truth), 19
isTRUE, 21

merge, 13, 14
merge.list, 12, 14
merge_dots_with_list, 13, 13

na, 7, 10, 14

parenthesise (parenthesize), 15
parenthesize, 15
print, 17
print.scalar_with_cause, 16
print.vector_with_cause
 (print.scalar_with_cause), 16
print_and_capture, 16

safe_deparse, 17
set_cause, 8, 18
setNames, 18
sQuote, 15
stop, 10
strip_attributes, 19

Truth, [19](#)
try, [10](#)

use_first, [23](#)

vapply, [6, 7](#)

warning, [10](#)