Package 'assertions'

June 2, 2023

Title Simple Assertions for Beautiful and Customisable Error Messages

Version 0.1.0

Description Provides simple assertions with sensible defaults and customisable error messages.
It offers convenient assertion call wrappers and a general assert function that can handle any condition.
Default error messages are user friendly and easily customized with inline code evaluation and styling powered by the 'cli' package.

License MIT + file LICENSE

Suggests covr, knitr, rmarkdown, shiny, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

Imports cli, glue, rlang

URL https://github.com/selkamand/assertions

VignetteBuilder knitr

Collate 'assert.R' 'assert_class.R' 'is_comparisons.R'
'is_functions.R' 'utils.R' 'assert_create.R' 'assert_type.R'
'assert_compare.R' 'assert_dataframe.R' 'assert_files.R'
'assert_functions.R' 'set_operations.R' 'assert_includes.R'
'assert_names.R' 'assert_numerical.R' 'assert_set.R'
'coverage_testing.R' 'export_testing.R' 'has.R'

NeedsCompilation no

Author Sam El-Kamand [aut, cre, cph] (<https://orcid.org/0000-0003-2270-8088>)

Maintainer Sam El-Kamand <sam.elkamand@gmail.com>

Repository CRAN

Date/Publication 2023-06-02 14:50:02 UTC

R topics documented:

assert *Assert that conditions are met*

Description

Assert that conditions are met

Usage

```
assert(..., msg = NULL, call = rlang::caller_env())
```

Arguments

...	a list of conditions to check
msg	A character string containing the error message to display if any of the conditions are not met. The string can include the placeholder failed_expressions to insert a list of the failed expressions. The string can also include ?s and ?is/are to insert the correct pluralization for the list of failed expressions.
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.

Value

invisible(TRUE) if all conditions are met, otherwise aborts with the error message specified by msg

Examples

```
try({
assert(1 == 1) # Passes
assert(2 == 2, 3 == 3) # Passes
assert(2 == 1, 3 == 3) # Throws default error
assert(2 == 1, 3 == 3, msg = "Custom error message") # Throws custom error
})
```

assertion_names *List assertion names*

Description

List all assertion names

Usage

```
assertion_names(exclude_create_and_chain = TRUE)
```

Arguments

`exclude_create_and_chain`
: exclude assert_create and assert_create_chain (flag)

Value

unique set of assertion names (character)

`assertion_tests` *Count tests per Assertion*

Description

Count the number of unit-tests per assertion. Note assertion_tests only finds tests where `expect_` and `assert_` are on the same line.

Usage

```
assertion_tests()
```

Value

two column data.frame describing assertion name and number of tests (expect_statement)

`assert_all_directories_exist` *Assert all files are directories*

Description

Assert that all paths supplied exist and are directories. To assert a single directory exists, see `assert_directory_exists()`

Usage

```
assert_all_directories_exist(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	Paths to directories (character)
msg	A character string containing the error message if file x is does not exist
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is exists and is a directory, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_directory(system.file("package = assertions")) # PASSES
assert_directory("foo") # Throws Error
})
```

---

assert_all_files_exist

Assert that all files exist

Description

Assert all files in vector exist. To assert a single file exists, see `assert_file_exists()`

Usage

```
assert_all_files_exist(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	Paths to files (character)
msg	A character string containing the error message if any files in x is does not exist
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if all files in x exist, otherwise aborts with the error message specified by msg

Examples

```
real_file <- system.file("DESCRIPTION", package = "assertions")

try({
assert_all_files_exist(c(real_file, real_file))
assert_all_files_exist(c("foo", "bar")) # Throws Error
})
```

---

assert_all_files_have_extension

Assert file extensions

Description

Assert that all filepaths supplied have one of the selected extensions. Does not require file to actually exist.

Usage

```
assert_all_files_have_extension(
  x,
  extensions,
  compression = FALSE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
extensions	valid extensions (character vector). Do not include the '.', e.g. supply extensions = 'txt' not extensions = '.txt'
compression	should compression extension '.gz', '.bz2' or '.xz' be removed first?
msg	A character string containing the error message if file x does not have the specified extensions
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x has any of the specified extensions, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_all_files_have_extension(c("foo.txt", "bar.txt"), extensions = "txt") # Passes
assert_all_files_have_extension(c("foo.txt", "bar.csv"), extensions = "csv") # Throws Error
})
```

---

assert_all_greater_than

Assert input is greater than a specified minimum value

Description

Assert all elements in a numeric vector/matrix are above some minimum value.

Usage

```
assert_all_greater_than(
  x,
  minimum,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object to check
minimum	The minimum value to compare against (number)
msg	A character string containing the error message to display if x is not greater than the specified minimum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is greater than the specified minimum value, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_all_greater_than(3, 2) # Passes
assert_all_greater_than(c(2,3,4), 1) # Passes
assert_all_greater_than(c(2,3,4), 2) # Passes
assert_all_greater_than(c(2,3,1), 3) # Throws default error
assert_all_greater_than(c(2,3,1), 3, msg = "custom error message") # Throws custom error
})
```

`assert_all_greater_than_or_equal_to` *Assert input is greater than or equal to a specified minimum value*

Description

Assert all elements in a numeric vector/matrix are above some minimum value.

Usage

```
assert_all_greater_than_or_equal_to(
  x,
  minimum,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	An object to check
`minimum`	The minimum value to compare against
`msg`	A character string containing the error message to display if `x` is not greater than or equal to the specified minimum value (string)
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` is greater than or equal to the specified minimum value, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_greater_than_or_equal_to(3, 2) # Passes
assert_greater_than_or_equal_to(c(3, 4, 5), 2) # Passes
assert_greater_than_or_equal_to(2, 3) # Throws error
})
```

`assert_character` *Assert input is a character vector*

Description

Assert an R object is a 'character' type. Works for **vector** and **matrix** objects. To assert an object is specifically a **character vector** see `assert_character_vector()`

Usage

```
assert_character(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

`x`	An object
`msg`	A character string containing the error message to display if `x` is not a character vector
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` is a character vector, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_character("a") # Passes
assert_character("a") # Passes
assert_character(c("a", "b", "c")) # Passes
assert_character(matrix(c('A', 'B', 'C', 'D')))  # Passes
assert_character(1:3) # Throws default error
assert_character(c("a", 1, "b"), "Custom error message") # Throws custom error
})
```

assert_character_vector *Assert input is a character vector*

Description

Assert an object is a character vector. Length 1 character vectors (strings) are considered vectors.

Usage

```
assert_character_vector(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a character vector
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a character vector, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_character_vector(c("a", "b", "c")) # Passes
assert_character_vector(c("a", 1, "b")) # Throws default error
assert_character_vector(matrix(c('A', 'B', 'C', 'D'))) # Throws error since type = matrix
assert_character_vector(c("a", 1, "b"), "Custom error message") # Throws custom error
})
```

assert_class *Assert object belongs to class*

Description

This function asserts that the input object belongs to `class`

Usage

```
assert_class(x, class, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

`x`	An input object
`class`	checks if `x` belongs to `class`. If multiple values of `class` are supplied, returns whether `x` belongs to any of them (character)
`msg`	A character string containing the error message to display if `x` does not belong to `class`
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` belongs to `class`, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_has_class(1, "numeric")  # Passes
assert_has_class(1, "character")  # Throws default error
})
```

assert_create *Create an assertion function*

Description

This function creates an assertion function that can be used to check the validity of an input. All assertions provided with this package are created using either `assert_create()` or `assert_create_chain()`

Usage

```
assert_create(func, default_error_msg = NULL)
```

Arguments

`func` A function defining the assertion criteria. This function should return a logical value (`TRUE` when assertion is passed or `FALSE` when it fails). Alternatively, instead of returning FALSE, you can return a string which will act as the error message. In this latter case, you don't need to supply a `default_error_msg`

`default_error_msg`
A character string providing an error message in case the assertion fails. Must be supplied if function `func` returns `FALSE` when assertion fails (as opposed to a string) Can include the following special terms

1. `{arg_name}` to refer to the name of the variable supplied to the assertion.
2. `{arg_value}` to refer to the value of the variable supplied to the assertion
3. `{code_to_evaluate}` to evaluate the code within the error message. Replace `code_to_evaluate` with your code
4. `{.strong bold_text}` to perform inline formatting. Replace `bold_text` with your text. See cli documentation for details

Value

An assertion function.

Examples

```
#' # Create an assertion function that checks that a character string is all
# lower case
assert_character <- assert_create(
  is.character,
  "{arg_name} must be a character vector, not a {class(arg_value)}"
)

# Use the assertion function
try({
is_lower("hello") # Returns invisible TRUE
is_lower("Hello") # Aborts the function with the error message
})
```

`assert_create_chain` *Create Chains of Assertions*

Description

Combine multiple assertion functions created by `assert_create()` into a single assertion function with diverse failure modes and error messages.

Usage

```
assert_create_chain(...)
```

Arguments

...	assertion functions created by `assert_create()`.

Value

A single assertion function that calls each of the input functions in the order they are supplied.

Examples

```
# Create an assertion function that checks for both positive integers and even values
assert_string <- assert_create_chain(
  assert_create(is.character, '{{arg_name}} must be a character'),
  assert_create(function(x){{ length(x)==1 }}, '{{arg_name}} must be length 1')
  )

# Use the assertion function to check a valid value
assert_string("String")

# Use the assertion function to check an invalid value
try({
assert_string(3)
# Output: Error: '3' must be a character
})
```

---

assert_dataframe *Assert input is a data frame*

Description

Assert input is a data frame

Usage

```
assert_dataframe(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a data frame
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a data frame, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_dataframe(mtcars) # Passes
assert_dataframe(data.frame()) # Passes

assert_dataframe(1:10) # Throws default error
assert_dataframe(matrix(1:6, 2, 3)) # Throws default error
assert_dataframe(c(1, 2, 3)) # Throws default error: "Error
assert_dataframe(list(a = 1, b = 2)) # Throws default error
assert_dataframe(factor(c(1, 2, 3))) # Throws default error

assert_dataframe(1:10, msg = "Custom error message") # Throws custom error
})
```

---

assert_directory_does_not_exist

Assert a directory does not exist

Description

Assert that a directory does not already exist. Useful for avoiding overwriting. This function is an exact copy of `assert_file_does_not_exist()` and included to make assertion code more readable.

Usage

```
assert_directory_does_not_exist(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	Path to a file (string)
`msg`	A character string containing the error message if file `x` already exists
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if directory `x` does not already exist, otherwise aborts with the error message specified by `msg`

Examples

```
real_dir <- system.file("tests", package = "assertions")

try({
assert_directory_does_not_exist("foo") # Passes
assert_directory_does_not_exist(real_dir) # Throws error
assert_directory_does_not_exist(c("foo", "bar")) # Throws Error (single file only)
})
```

---

assert_directory_exists

Assert are directory exists

Description

Assert a directory exists. To assert all directories in a vector exist, see `assert_all_directories_exist()`

Usage

```
assert_directory_exists(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	Path to a directory (string)
`msg`	A character string containing the error message if file `x` is does not exist
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` is exists and is a directory, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_directory_exists(system.file("package = assertions")) # PASS
assert_all_directories_exist("foo") # Throws Error
})
```

assert_equal *Assert that the input objects are equal*

Description

Is x equal to y. powered by the `all.equal()` function.

Usage

```
assert_equal(
  x,
  y,
  tolerance = sqrt(.Machine$double.eps),
  check_names = TRUE,
  check_environment = TRUE,
  check_tzone = TRUE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object to check
y	The value to compare against
tolerance	Differences smaller than tolerance are not reported. The default value is close to 1.5e-8 (numeric >= 0).
check_names	should the names(.) of target and current should be compare (flag)
check_environment	should the environments of functions should be compared? You may need to set check.environment=FALSE in unexpected cases, such as when comparing two nls() fits. (flag)
check_tzone	should "tzone" attributes be compared. Important for comparing POSIXt objects. (flag)
msg	A character string containing the error message to display if x is not equal to y
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is equal to the specified value, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_equal(3, 3) # Passes
assert_equal(c(3, 3, 3), 3, ) # Fails
assert_equal(2, 3) # Throws error
})
```

`assert_excludes` *Assert object does not include any illegal values*

Description

Assert x does not include illegal elements

Usage

```
assert_excludes(
  x,
  illegal,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	An object
`illegal`	The prohibited elements to check for
`msg`	A character string describing the error message if `x` includes any `illegal` elements
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` includes any `illegal` elements, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_directory(system.file("package = assertions"))
assert_directory("foo") # Throws Error
})
```

assert_factor_vector *Assert input is a factor*

Description

Assert an R object is a factor. Note that no assert_factor function exists since in R factors are always vector quantities (never scalar / in matrices)

Usage

```
assert_factor_vector(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a factor
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Details

Technically this function name is misleading, since `is.vector(factor(1)) == FALSE` but since they act exactly like vectors to end users, I think this name is the most suitable

Value

invisible(TRUE) if x is a factor, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_factor_vector(factor(c("a", "b", "c"))) # Passes
assert_factor_vector(c("a", "b", "c")) # Throws default error
assert_factor_vector(factor(c("a", "b", "c")), "Custom error message") # Passes
assert_factor_vector(1:3, "Custom error message") # Throws custom error
})
```

`assert_file_does_not_exist` *Assert a file does not exist*

Description

Assert that a file does not exist. Useful for avoiding overwriting.

Usage

```
assert_file_does_not_exist(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	Path to a file (string)
`msg`	A character string containing the error message if file `x` already exists
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if file `x` does not exist, otherwise aborts with the error message specified by `msg`

Examples

```
real_file <- system.file("DESCRIPTION", package = "assertions")

try({
assert_file_does_not_exist("foo") # Passes
assert_file_does_not_exist(real_file) # Throws error
assert_file_does_not_exist(c("foo", "bar")) # Throws Error (single file only)
})
```

assert_file_exists *Assert a file exists*

Description

Assert that a file exists. To assert all files in a vector exist, see `assert_all_files_exist()`

Usage

```
assert_file_exists(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	Path to a file (string)
msg	A character string containing the error message if file x is does not exist
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if file x exists, otherwise aborts with the error message specified by msg

Examples

```
real_file <- system.file("DESCRIPTION", package = "assertions")

try({
assert_file_exists(real_file) # PASSES
assert_file_exists("foo") # Throws Error
assert_file_exists(c(real_file, real_file)) # Throws Error (should use assert_all_files_exist)

})
```

assert_file_has_extension *Assert file extensions*

Description

Assert that a filepath includes one of the selected extensions. Does not require file to actually exist.

Usage

```
assert_file_has_extension(
  x,
  extensions,
  compression = FALSE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
extensions	valid extensions (character vector). Do not include the '.', e.g. supply `extensions = 'txt'` not `extensions = '.txt'`
compression	should compression extension '.gz', '.bz2' or '.xz' be removed first?
msg	A character string containing the error message if file x does not have the specified extensions
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x has any of the specified extensions, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_file_has_extension("foo.txt", extensions = "txt") # Passes
assert_file_has_extension("file.txt", extensions = "csv") # Throws Error
})
```

---

assert_flag *Assert input is a scalar logical*

Description

Assert input is a flag (a logical of length 1: TRUE or FALSE)

Usage

```
assert_flag(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a scalar logical
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a scalar logical, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_flag(TRUE) # Passes
assert_flag(FALSE) # Passes
assert_flag(c(TRUE, FALSE)) # Throws default error
assert_flag(1, "Custom error message") # Throws custom error
})
```

---

assert_function *Assert input is a function*

Description

Assert input is a function

Usage

```
assert_function(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a function
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a function, otherwise aborts with the error message specified by msg

Examples

```
try({
# Assert that a variable is a function
x <- function(a, b) { a + b }
assert_function(x)  # does nothing

# Assert that a variable is not a function
x <- "not a function"
assert_function(x)  # stops execution and prints an error message
})
```

`assert_function_expects_n_arguments`
Assert function expects n arguments

Description

Assert a function expects n arguments, with user control over how variable arguments (...) are counted (default throws error)

Usage

```
assert_function_expects_n_arguments(
  x,
  n,
  dots = c("throw_error", "count_as_0", "count_as_1", "count_as_inf"),
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	a function to check has exactly N arguments
`n`	number of arguments that must be expected by function to pass assertion (integer)
`dots`	how to deal with '...' dots (a.k.a variable arguments). Should we count as 0, 1 or infinite arguments. Or, do we just throw an error when we see '...' (default)
`msg`	The error message thrown if the assertion fails (string)
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if function x expects exactly n arguments, otherwise aborts with the error message specified by msg

---

`assert_greater_than` *Assert input is greater than some minimum value*

Description

Assert a number is greater than a specified minimum value. To check all numbers in a vector / matrix are above a minimum value, see `assert_all_greater_than()`

Usage

```
assert_greater_than(
  x,
  minimum,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object to check
minimum	The minimum value to compare against (number)
msg	A character string containing the error message to display if x is not greater than the specified minimum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is greater than the specified minimum value, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_greater_than(3, 2) # Passes
assert_greater_than(3, 2) # Passes
assert_greater_than(c(2,3,4), 1) # Throws error (Must be a number)
assert_greater_than('A', 1) # Throws error (Must be a number)
assert_greater_than(2, 3, msg = "custom error message") # Throws custom error
})
```

assert_greater_than_or_equal_to *Assert input is greater than or equal to a specified minimum value*

Description

Assert all elements in a numeric vector/matrix are above or equal to some minimum value. For vectorized version see assert_all_greater_than_or_equal_to()

Usage

```
assert_greater_than_or_equal_to(
  x,
  minimum,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object to check
minimum	The minimum value to compare against
msg	A character string containing the error message to display if x is not greater than or equal to the specified minimum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is greater than or equal to the specified minimum value, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_greater_than_or_equal_to(3, 2) # Passes
assert_greater_than_or_equal_to(c(3, 4, 5), 2) # Throws error
assert_greater_than_or_equal_to(2, 3) # Throws error
})
```

assert_identical *Assert that the input object is identical to a specified value*

Description

Assert that the input object is identical to a specified value

Usage

```
assert_identical(x, y, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object to check
y	The value to compare against
msg	A character string containing the error message to display if x is not identical to the specified value
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is identical to the specified value, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_identical(3, 3) # Passes
assert_identical(c(3, 3, 3), 3) # Throws error
assert_identical(2, 3) # Throws error
})
```

assert_includes *Assert object includes required*

Description

Assert x includes required elements

Usage

```
assert_includes(
  x,
  required,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
required	The required elements to check for
msg	A character string describing the error message if x does not include required elements
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x includes all required elements, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_directory(system.file("package = assertions"))
assert_directory("foo") # Throws Error
})
```

---

assert_int *Assert input is an integer*

Description

Assert input is an integer

Usage

```
assert_int(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not an integer
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is an integer, otherwise aborts with the error message specified by msg

Note

In R, integers are whole numbers. Both integers and doubles (numbers with decimals) are considered numeric. This function checks that x specifically belong to the integer class.

Examples

```
try({
assert_int(1) # Passes
assert_int(1:10) # Passes
assert_int(c(1, 2, 3)) # Passes
assert_int("a") # Throws default error
assert_int(1.5, msg = "Custom error message") # Throws custom error
})
```

---

assert_list *Assert input is a list*

Description

Assert input is a list

Usage

```
assert_list(
  x,
  include_dataframes = FALSE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
include_dataframes	A logical indicating whether data_frames should be considered vectors. Default is FALSE.
msg	A character string containing the error message to display if x is not a list
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a list, otherwise aborts with the error message specified by msg

Examples

```
try({
# Assert that a variable is a list
x <- list(1, 2, 3)
assert_list(x)  # does nothing

# Assert that a variable is not a list
x <- "not a list"
assert_list(x)  # stops execution and prints an error message
})
```

---

assert_logical *Assert input is logical*

Description

Assert an R object is 'logical' (TRUE/FALSE). Works for **vector** and **matrix** objects. To assert an object is specifically a **logical vector** see `assert_logical_vector()`

Usage

```
assert_logical(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not logical
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is logical, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_logical(TRUE) # Passes
assert_logical(c(TRUE, FALSE, TRUE)) # Passes
assert_logical(c("a", "b")) # Throws default error
assert_logical(1:3, "Custom error message") # Throws custom error
})
```

---

assert_logical_vector *Assert input is an atomic logical vector*

Description

Assert input is an atomic logical vector

Usage

```
assert_logical_vector(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not an atomic logical vector
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is an atomic logical vector, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_logical_vector(c(TRUE, TRUE, TRUE)) # Passes
assert_logical_vector("a") # Throws default error
assert_logical_vector(c(1, 0, 1), "Custom error message") # Throws custom error
})
```

`assert_matrix` *Assert input is a matrix*

Description

Assert input is a matrix

Usage

```
assert_matrix(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

`x`	An object
`msg`	A character string containing the error message to display if `x` is not a matrix
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` is a matrix, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_matrix(matrix(1:9, 3)) # Passes
assert_matrix(matrix(1:9, 3, 3)) # Passes
assert_matrix(c(1, 2, 3)) # Throws default error
assert_matrix(1:10, "Custom error message") # Throws custom error
})
```

`assert_names_include` *Assert that the input object includes a specified name*

Description

Assert that the input object includes a specified name

Usage

```
assert_names_include(
  x,
  names,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	An object to check for the presence of specific names
`names`	A character vector of names to check for in `x`
`msg`	A character string containing the error message to display if any of the `names` are not present in `x`
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if all `names` are present in `x`, otherwise aborts with the error message specified by `msg`

Examples

```
try({
x <- list(a = 1, b = 2, c = 3)

assert_includes_name(x, "a") # Passes
assert_includes_name(x, c("a", "b")) # Passes
assert_includes_name(x, c("a", "b", "d")) # Throws default error message

assert_includes_name(x, c("a", "b", "d"), "Custom error message") # Throws custom error message
})
```

assert_non_empty_string *Assert input is a non empty character string*

Description

Asserts input is a string, and nonempty (i.e. not equal to '')

Usage

```
assert_non_empty_string(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a character vector, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_non_empty_string("a") # Passes
assert_non_empty_string("") # Fails
})
```

`assert_no_duplicates` *Assert that the input vector has no duplicates*

Description

Assert the input vector has no duplicated elements

Usage

```
assert_no_duplicates(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

`x`	A vector.
`msg`	A character string containing the error message to display if `x` has duplicates.
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` has no duplicates, otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_no_duplicates(c(1, 2, 3))  # Passes
assert_no_duplicates(c(1, 2, 2))  # Throws default error

assert_no_duplicates(c(1, 2, 3), msg = "Custom error message")  # Passes
assert_no_duplicates(c(1, 2, 2), msg = "Custom error message")  # Throws custom error
})
```

`assert_no_missing` *Assert that the input vector has no missing values*

Description

This function asserts that the input vector has no missing values (`NA`) and aborts with an error message if it does.

Usage

```
assert_no_missing(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

`x`	A vector.
`msg`	A character string containing the error message to display if `x` has missing values.
`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if `x` has no missing values (NA), otherwise aborts with the error message specified by `msg`

Examples

```
try({
assert_no_missing(c(1, 2, 3))  # Passes
assert_no_missing(c(1, NA, 2))  # Throws default error

assert_no_missing(c(1, 2, 3), msg = "Custom error message")  # Passes
assert_no_missing(c(1, NA, 2), msg = "Custom error message")  # Throws custom error
})
```

assert_number *Assert input is a number*

Description

A number is a length 1 numeric vector. Numbers can be either integers or doubles.

Usage

```
assert_number(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a number
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a number, otherwise aborts with the error message specified by msg

Examples

```
assert_number(2) # Passes
try({
assert_number(c(2, 3)) # Throws default error
assert_number("a") # Throws default error
assert_number(c("a", 1, "b"), "Custom error message") # Throws custom error
})
```

assert_numeric *Assert input is numeric*

Description

Assert an R object is numeric Works for **vector** and **matrix** objects. To assert an object is specifically a **numeric vector** see `assert_numeric_vector()`

Usage

```
assert_numeric(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not numeric
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is numeric, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_numeric(1:3) # Passes
assert_numeric(1.5:5.5) # Passes
assert_numeric(c("a", "b", "c")) # Throws default error
assert_numeric(c("a", 1, "b"), "Custom error message") # Throws custom error
})
```

---

assert_numeric_vector *Assert input is a numeric vector*

Description

Assert input is a numeric vector

Usage

```
assert_numeric_vector(
  x,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a numeric vector
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a numeric vector, otherwise aborts with the error message specified by msg

---

`assert_reactive` *Assert that x is reactive*

Description

Assert that x is reactive

Usage

```
assert_reactive(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not reactive
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a reactive, otherwise aborts with the error message specified by msg

Examples

```
try({
# Assert that a variable is reactive
x <- shiny::reactive(1)
assert_reactive(x)  # does nothing

# Assert that a variable is not a list
x <- 1
assert_reactive(x)  # stops execution and prints an error message
})
```

assert_string *Assert input is a character string*

Description

Assert input is a character string

Usage

```
assert_string(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a string
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a string, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_string("a") # Passes
assert_string(c("a", "b", "c")) # Throws default error
assert_string(1:3) # Throws default error
assert_string(c("a", 1, "b"), "Custom error message") # Throws custom error
})
```

assert_subset *Check if a vector is a subset of another*

Description

This function checks that x is a subset of y

Usage

```
assert_subset(x, y, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	A vector to check
y	the acceptible values that x can take
msg	The error message thrown if the assertion fails (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

Returns invisible(TRUE) if x is a subset of y, otherwise throws an error

Examples

```
try({
assert_subset(1:3, 1:5) # Passes
assert_subset(c("A", "B", "C"), c("A", "B")) # Throws error since "C" is not present in first vector
})
```

---

assert_vector *Assert input is a vector*

Description

Assert input is a vector

Usage

```
assert_vector(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	A character string containing the error message to display if x is not a vector
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a vector, otherwise aborts with the error message specified by msg

Note

By default, lists are not considered vectors (i.e. `include_lists = FALSE`) to align with what end-users will expect, in spite of these objects technically being vectors.

Examples

```
try({
assert_vector(c(1, 2, 3)) # Passes
assert_vector(matrix(1:6, 2, 3)) # Throws default error message
assert_vector(1:3) # Passes

assert_vector(list(1, 2, 3)) # Throws default error message
assert_vector(list(1, 2, 3), include_lists = TRUE) # Passes

assert_vector(c("a", 1, "b"), "Custom error message") # Throws custom error message
assert_vector(factor(c(1, 2, 3)), "Custom error message") # Throws custom error message
})
```

---

assert_whole_number *Assert that the input object is a whole number*

Description

Check if x is a whole number (no decimal)

Usage

```
assert_whole_number(x, msg = NULL, call = rlang::caller_env(), arg_name = NULL)
```

Arguments

x	An object
msg	The error message thrown if the assertion fails (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

Value

invisible(TRUE) if x is a whole number, otherwise aborts with the error message specified by msg

Examples

```
try({
assert_whole_number(24) # Passes
assert_whole_number(2.5) # Throws error
})
```

`check_all_assertions_are_tested_enough` *Check assertions are tested enough*

Description

Check assertions are tested enough

Usage

```
check_all_assertions_are_tested_enough(min_required_tests = 5)
```

Arguments

`min_required_tests`
: min number of tests (expect statements) per assertion

Value

TRUE if all assertions sufficiently tested. Otherwise throws error

---

`common_roxygen_params` *Common Parameter Descriptions*

Description

Common Parameter Descriptions

Usage

```
common_roxygen_params(call, arg_name, msg, ...)
```

Arguments

`call`	Only relevant when pooling assertions into multi-assertion helper functions. See cli_abort for details.
`arg_name`	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).
`msg`	The error message thrown if the assertion fails (string)
`...`	Used to pass any arguments to assertion function

excludes_advanced *Check if an object does not contain prohibited elements*

Description

This function checks that x does not include any of the illegal elements. x must be the same type as illegal.

Usage

```
excludes_advanced(x, illegal)
```

Arguments

x	An object to check
illegal	The prohibited elements to check for

Value

Returns TRUE if x is the same type as illegal and x does not include any of the illegal elements. Otherwise returns a string representing the appropriate error message to display

format_as_bullets *Preprocess character vectors for cli::cli_abort()*

Description

The format_as_bullets function is used for preprocessing character vectors by adding names. These names are used to denote bullet points when the character vector is passed to cli::cli_abort(). This allows for the easy creation of bullet point lists in error messages. The bullet argument allows the user to specify the desired bullet point symbol. The default bullet point symbols are: *, >, , x, v, i, and !.

Usage

```
format_as_bullets(x, bullet = c("*", ">", " ", "x", "v", "i", "!"))
```

Arguments

x	A list of character strings
bullet	One of ", '>', ' ', 'x', 'v', 'i', '!' (default: ") The character to use as the bullet point for each element of x.

Value

A character string with each element of x formatted as a bullet point

format_inline *Preprocess character vectors for cli package functions*

Description

Preprocess character vectors for cli package functions

Usage

```
format_inline(x, inline_tag = c("strong", "emph", "code", "arg"))
```

Arguments

x	A character vector
inline_tag	A character vector of inline tag names (e.g. "strong", "emph", "code", "arg")

Value

A character vector with inline tags applied to each element

has_all_names *Check if a named object has all specified names*

Description

This function returns a logical value indicating whether the object x has all the names specified in names.

Usage

```
has_all_names(x, names)
```

Arguments

x	a named object
names	A character vector of names to check for in x.

Value

A logical value indicating whether x has all the names specified in names

has_class *Check object is some class*

Description

This function checks whether object is a specific class

Usage

```
has_class(x, class)
```

Arguments

x	A value to check.
class	checks if x belongs to class. If multiple values of class are supplied, returns whether x belongs to any of them (character)

Value

A logical scalar indicating x belongs to class

Examples

```
if(interactive()) {
  has_class(1, "numeric") # TRUE
  has_class(1, "character") # FALSE
}
```

has_duplicates *Check if a vector has duplicates*

Description

This function returns a logical value indicating whether the input vector contains duplicated elements.

Usage

```
has_duplicates(x)
```

Arguments

x	A vector.

Value

A logical value indicating whether the input vector contains duplicated elements.

Examples

```
if(interactive()){
has_duplicates(c(1, 2, 3))  # returns FALSE
has_duplicates(c(1, 2, 2))  # returns TRUE
}
```

`has_extension` *Title*

Description

Title

Usage

```
has_extension(x, extensions, compression = FALSE)
```

Arguments

`x`	object to test
`extensions`	valid extensions (character vector). Do not include the '.', e.g. supply `extensions = 'txt'` not `extensions = '.txt'`
`compression`	should compression extension '.gz', '.bz2' or '.xz' be removed first?

Value

TRUE if all x have valid extensions as supplied by `extensions` (flag)

`has_missing_values` *Check if a vector has missing values*

Description

This function returns a logical value indicating whether the input vector contains missing values (`NA`).

Usage

```
has_missing_values(x)
```

Arguments

`x`	A vector.

Value

A logical value indicating whether the input vector contains missing values.

Examples

```
if(interactive()){
has_missing_values(c(1, 2, 3))  # returns FALSE
has_missing_values(c(1, NA, 2))  # returns TRUE
}
```

`has_no_duplicates` *Check if a vector has no duplicates*

Description

This function returns a logical value indicating whether the input vector contains no duplicated elements.

Usage

```
has_no_duplicates(x)
```

Arguments

x	A vector.

Value

A logical value indicating whether the input vector contains no duplicated elements.

Examples

```
if(interactive()){
has_no_duplicates(c(1, 2, 3))  # returns TRUE
has_no_duplicates(c(1, 2, 2))  # returns FALSE
}
```

has_no_missing_values *Check if a vector has no missing values*

Description

This function returns a logical value indicating whether the input vector contains no missing values (NA).

Usage

```
has_no_missing_values(x)
```

Arguments

x	A vector.

Value

A logical value indicating whether the input vector contains no missing values.

Examples

```
if(interactive()){
has_no_missing_values(c(1, 2, 3))  # returns TRUE
has_no_missing_values(c(1, NA, 2))  # returns FALSE
}
```

includes *Check if All Values in Required are in x*

Description

Checks if all elements of required are present in x.

Usage

```
includes(x, required)
```

Arguments

x	A vector of elements.
required	A vector of elements to check for inclusion in x.

Value

A logical value indicating whether all elements of required are present in x (TRUE) or not (FALSE).

includes_advanced *Check if an object contains required elements*

Description

This function checks that `x` includes all of the `required` elements. `x` must be the same type as `required`.

Usage

```
includes_advanced(x, required)
```

Arguments

`x`	An object to check
`required`	The required elements to check for

Value

Returns TRUE if `x` is the same type as `required` and `x` includes all the `required` elements. Otherwise returns a string representing the appropriate error message to display

is_character_vector *Check if an object is a character vector*

Description

Check if an object is a character vector

Usage

```
is_character_vector(x)
```

Arguments

`x`	An object to check.

Value

A logical value indicating whether `x` is a character vector.

`is_equal` *Check equality of two objects*

Description

Is x equal to y. powered by the `all.equal()` function.

Usage

```
is_equal(
  x,
  y,
  tolerance = sqrt(.Machine$double.eps),
  check_names = TRUE,
  check_environment = TRUE,
  check_tzone = TRUE
)
```

Arguments

x	first object to compare
y	second object to compare
tolerance	Differences smaller than tolerance are not reported. The default value is close to 1.5e-8 (numeric >= 0).
check_names	should the names(.) of target and current should be compare (flag)
check_environment	should the environments of functions should be compared? You may need to set check.environment=FALSE in unexpected cases, such as when comparing two nls() fits. (flag)
check_tzone	should "tzone" attributes be compared. Important for comparing POSIXt objects. (flag)

Value

TRUE if x is equal to y

Examples

```
if(interactive()){
is_equal(1, 1) #TRUE
is_equal(c(1, 2), 1) #FALSE

is_equal(c("A", "B"), c("A", "B")) #TRUE
is_equal("A", "B") #FALSE
}
```

is_flag *Check if a value is a logical flag*

Description

This function checks if a value is a logical scalar (i.e., a single logical value).

Usage

```
is_flag(x)
```

Arguments

x	A value to check.

Value

A logical scalar indicating whether x is a logical flag.

is_flag_advanced *Check if x is a flag*

Description

This function is designed for use with `assert_create_advanced`. It must return TRUE for the assertion to pass or a string representing the error message if the assertion should fail.

Usage

```
is_flag_advanced(x)
```

Arguments

x	A value to be checked

Value

Returns invisible(TRUE) if x is a logical value with length 1. Returns a string with an error message if x is not a logical value or has a length other than 1.

is_greater_than *Check if a numeric vector is greater than a specified minimum value*

Description

This function checks if a numeric vector is greater than a specified minimum value. It can also optionally check if all elements of the vector must be greater than the minimum value or if only one element is sufficient

Usage

```
is_greater_than(x, minimum)
```

Arguments

x	a numeric vector to check
minimum	The minimum value to compare against

Value

A logical value indicating whether all elements of the numeric vector x are greater than the specified minimum value

Examples

```
if(interactive()){
is_greater_than(c(2,3,4), 1) # TRUE
is_greater_than(c(2,3,4), 2) # TRUE
is_greater_than(c(2,3,1), 3) # FALSE
}
```

is_greater_than_or_equal_to *Check if a numeric vector is greater than or equal to a specified minimum value*

Description

This function checks if a numeric vector is greater than or equal to a specified minimum value. It can also optionally check if all elements of the vector must be greater than or equal to the minimum value or if only one element is sufficient

Usage

```
is_greater_than_or_equal_to(x, minimum)
```

Arguments

x	a numeric vector to check
minimum	The minimum value to compare against

Value

A logical value indicating whether all elements of the numeric vector x are greater than or equal to the specified minimum value

Examples

```
if(interactive()){
is_greater_than_or_equal_to(c(2,3,4), 1) # TRUE
is_greater_than_or_equal_to(c(2,3,4), 2) # TRUE
is_greater_than_or_equal_to(c(2,3,1), 3) # FALSE
}
```

---

is_identical *Check if two objects are identical*

Description

Check if two objects are identical

Usage

```
is_identical(x, y)
```

Arguments

x	first object to compare
y	second object to compare

Value

logical value indicating whether or not the objects are identical

is_list *Check if a value is a list*

Description

This function checks if a value is a list. By default, definition of a 'list' excludes data.frames in spite of them technically being lists. This behaviour can be changed by setting `include_dataframes = TRUE`

Usage

```
is_list(x, include_dataframes = FALSE)
```

Arguments

`x`	A value to check.
`include_dataframes`	A logical indicating whether data_frames should be considered vectors. Default is `FALSE`.

Value

A logical scalar indicating whether `x` is a list.

Examples

```
if(interactive()){
is_list(list(1, 2)) # TRUE
is_list(c(1, 2, 3)) # FALSE
is_list(data.frame()) # FALSE
is_list(data.frame(), include_dataframes = TRUE) # TRUE
}
```

is_logical_vector *Check if an object is a logical vector*

Description

Check if an object is a logical vector

Usage

```
is_logical_vector(x)
```

Arguments

`x`	An object to check.

Value

A logical value indicating whether x is a logical vector.

`is_non_empty_string_advanced` *Check if x is a nonempty string*

Description

This function is designed for use with `assert_create`. It returns TRUE for the assertion to pass or a string representing the error message if the assertion should fail.

Usage

```
is_non_empty_string_advanced(x)
```

Arguments

x	A value to be checked

Value

Returns invisible(TRUE) if x is a character value with length 1 and at least 1 character in string. Returns a string with an error message if x is not a character value or has a length other than 1.

`is_number` *Check if an object is a single number*

Description

Check if an object is a single number

Usage

```
is_number(x)
```

Arguments

x	An object to check.

Value

A logical value indicating whether x is a single number.

is_number_advanced *Check if x is a number*

Description

This function is designed for use with `assert_create_advanced`. It must return TRUE for the assertion to pass or a string representing the error message if the assertion should fail.

Usage

```
is_number_advanced(x)
```

Arguments

x	A value to be checked

Value

Returns invisible(TRUE) if x is a numeric value with length 1. Returns a string with an error message if x is not a numeric value or has a length other than 1.

is_numeric_vector *Check if an object is a numeric vector*

Description

This function checks if an object is a numeric vector in R.

Usage

```
is_numeric_vector(x)
```

Arguments

x	An object to check.

Value

A logical value indicating whether x is a numeric vector.

Examples

```
if(interactive()){
is_numeric_vector(c(1, 2, 3)) # TRUE
is_numeric_vector(list(1, 2, 3)) # FALSE
is_numeric_vector(1:5) # TRUE
is_numeric_vector("hello") # FALSE
is_numeric_vector(list(1, 2, "a")) # FALSE
}
```

---

is_reactive *Check if a value is reactive*

Description

This function checks if a value is reactive

Usage

```
is_reactive(x)
```

Arguments

x	A value to check.

Value

A logical scalar indicating whether x is a list.

Examples

```
if(interactive()){
is_reactive(shiny::reactive(1)) # TRUE
is_reactive(1) # FALSE
}
```

is_same_type *Check equality of type*

Description

Is type of x the same as y (according to typof)

Usage

```
is_same_type(x, y)
```

Arguments

x	first object to compare
y	second object to compare

Value

TRUE if x and y are of the same type, otherwise FALSE

is_string *Check if an object is a single string*

Description

Check if an object is a single string

Usage

```
is_string(x)
```

Arguments

x	An object to check.

Value

A logical value indicating whether x is a single string.

is_string_advanced *Check if x is a string*

Description

This function is designed for use with `assert_create`. It returns TRUE for the assertion to pass or a string representing the error message if the assertion should fail.

Usage

```
is_string_advanced(x)
```

Arguments

x	A value to be checked

Value

Returns invisible(TRUE) if x is a character value with length 1. Returns a string with an error message if x is not a character value or has a length other than 1.

is_subset *Check if one set is a subset of another*

Description

Determines if all elements in set `x` are also present in set `y`.

Usage

```
is_subset(x, y)
```

Arguments

x	A numeric, character, or logical vector.
y	A numeric, character, or logical vector.

Value

A logical value indicating whether `x` is a subset of `y`.

is_superset *Check if one set is a superset of another*

Description

Determines if all elements in set y are also present in set x.

Usage

```
is_superset(x, y)
```

Arguments

x	A numeric, character, or logical vector.
y	A numeric, character, or logical vector.

Value

A logical value indicating whether x is a superset of y.

is_vector *Check if an object is a vector This function checks if an object is a vector*

Description

Check if an object is a vector This function checks if an object is a vector

Usage

```
is_vector(x)
```

Arguments

x	An object to check

Value

A logical indicating whether x is a vector

setopts_are_equal *Compare Sets for Equality*

Description

Determine if the two sets are equal.

Usage

```
setopts_are_equal(x, y)
```

Arguments

x	A vector of elements.
y	A vector of elements.

Value

A logical value indicating whether the sets are equal (TRUE) or not (FALSE).

setopts_common_elements *Find Common Elements*

Description

Find the elements that are present in both sets.

Usage

```
setopts_common_elements(x, y)
```

Arguments

x	A vector of elements.
y	A vector of elements.

Value

A vector of elements that are present in both sets.

`setopts_count_exlusive_to_first`
Count of Elements Exclusive to First Set

Description

Counts the number of elements that are in the first set but not in the second set.

Usage

```
setopts_count_exlusive_to_first(x, y)
```

Arguments

x	A vector of elements.
y	A vector of elements.

Value

A scalar representing the number of elements that are in the first set but not in the second set.

`setopts_exlusive_to_first`
Elements Exclusive to First Set

Description

Finds the elements that are in the first set but not in the second set.

Usage

```
setopts_exlusive_to_first(x, y)
```

Arguments

x	A vector of elements.
y	A vector of elements.

Value

A vector of elements that are in the first set but not in the second set.

util_count_duplicates *Count the number of duplicated values in a vector*

Description

This function returns the number of duplicated values in the input vector.

Usage

```
util_count_duplicates(x)
```

Arguments

x	A vector.

Value

The number of duplicated values in the input vector.

Examples

```
if(interactive()) {
util_count_duplicates(c(1, 2, 2))  # returns 1
util_count_duplicates(c(1, 2, 3))  # returns 0
}
```

util_count_missing *Count the number of missing values in a vector*

Description

This function returns the number of missing values (NA) in the input vector.

Usage

```
util_count_missing(x)
```

Arguments

x	A vector.

Value

The number of missing values in the input vector.

Examples

```
if(interactive()){
util_count_missing(c(1, 2, 3))  # returns 0
util_count_missing(c(1, NA, 2))  # returns 1
}
```

`util_get_duplicated_values` *Get the duplicated values in a vector*

Description

This function returns a vector of the duplicated values in the input vector.

Usage

```
util_get_duplicated_values(x)
```

Arguments

x	A vector.

Value

A vector of the duplicated values in the input vector.

Examples

```
if(interactive()) {
util_get_duplicated_values(c(1, 2, 2))  # returns 2
util_get_duplicated_values(c(1, 2, 3))  # returns NULL
}
```

Index