

# Package ‘assertions’

May 7, 2026

**Title** Simple Assertions for Beautiful and Customisable Error Messages

**Version** 0.3.0

**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** 7.3.3

**Imports** cli, glue, methods, rlang

**URL** <https://github.com/selkamand/assertions>,  
<https://selkamand.github.io/assertions/>

**VignetteBuilder** knitr

**Collate** 'assert.R' 'assert\_class.R' 'is\_comparisons.R'  
'is\_functions.R' 'utils.R' 'assert\_create.R' 'has.R'  
'assert\_type.R' 'assert\_compare.R' 'assert\_dataframe.R'  
'assert\_files.R' 'assert\_functions.R' 'set\_operations.R'  
'assert\_includes.R' 'assert\_length.R' 'assert\_names.R'  
'assert\_null.R' 'assert\_numerical.R' 'assert\_packages.R'  
'assert\_regex.R' 'assert\_set.R' 'coverage\_testing.R'  
'export\_testing.R'

**BugReports** <https://github.com/selkamand/assertions/issues>

**NeedsCompilation** no

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

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

**Repository** CRAN

**Date/Publication** 2026-02-13 12:50:10 UTC

## Contents

assert	4
assertion_names	5
assertion_tests	5
assert_all_between	6
assert_all_directories_exist	7
assert_all_files_exist	8
assert_all_files_have_extension	9
assert_all_finite	10
assert_all_greater_than	10
assert_all_greater_than_or_equal_to	11
assert_all_less_than	12
assert_all_less_than_or_equal_to	13
assert_all_strings_contain	14
assert_between	15
assert_character	17
assert_character_vector	18
assert_character_vector_or_glue	19
assert_class	20
assert_connection	20
assert_create	21
assert_create_chain	22
assert_dataframe	23
assert_directory_does_not_exist	24
assert_directory_exists	25
assert_equal	26
assert_excludes	27
assert_factor_vector	28
assert_file_does_not_exist	29
assert_file_exists	30
assert_file_has_extension	30
assert_finite	31
assert_flag	32
assert_function	33
assert_function_expects	34
assert_function_expects_n_arguments	35
assert_function_variadic	36
assert_greater_than	37
assert_greater_than_or_equal_to	38
assert_identical	39
assert_includes	39
assert_int	40
assert_length	41
assert_length_greater_than	42
assert_length_greater_than_or_equal_to	43
assert_length_less_than	43
assert_length_less_than_or_equal_to	44

assert_less_than . . . . .	45
assert_less_than_or_equal_to . . . . .	46
assert_list . . . . .	47
assert_logical . . . . .	48
assert_logical_vector . . . . .	48
assert_matrix . . . . .	49
assert_names_include . . . . .	50
assert_non_empty_string . . . . .	51
assert_non_null . . . . .	52
assert_no_duplicates . . . . .	53
assert_no_missing . . . . .	54
assert_null . . . . .	55
assert_number . . . . .	55
assert_numeric . . . . .	56
assert_numeric_vector . . . . .	57
assert_one_of . . . . .	58
assert_packages_installed . . . . .	58
assert_reactive . . . . .	59
assert_scalar . . . . .	60
assert_set_equal . . . . .	61
assert_string . . . . .	62
assert_string_contains . . . . .	63
assert_subset . . . . .	64
assert_vector . . . . .	64
assert_whole_number . . . . .	65
check_all_assertions_are_tested_enough . . . . .	66
common_roxygen_params . . . . .	67
contains_pattern . . . . .	67
excludes_advanced . . . . .	68
format_as_bullets . . . . .	68
format_inline . . . . .	69
has_all_names . . . . .	69
has_class . . . . .	70
has_duplicates . . . . .	70
has_extension . . . . .	71
has_missing_values . . . . .	71
has_no_duplicates . . . . .	72
has_no_missing_values . . . . .	73
includes . . . . .	73
includes_advanced . . . . .	74
is_between . . . . .	74
is_character_vector . . . . .	75
is_character_vector_or_glue . . . . .	75
is_equal . . . . .	76
is_flag . . . . .	77
is_flag_advanced . . . . .	77
is_greater_than . . . . .	78
is_greater_than_or_equal_to . . . . .	78

is_identical . . . . .	79
is_less_than . . . . .	80
is_less_than_or_equal_to . . . . .	80
is_list . . . . .	81
is_logical_vector . . . . .	82
is_non_empty_string_advanced . . . . .	82
is_number . . . . .	83
is_number_advanced . . . . .	83
is_numeric_vector . . . . .	84
is_reactive . . . . .	84
is_same_type . . . . .	85
is_string . . . . .	85
is_string_advanced . . . . .	86
is_subset . . . . .	86
is_superset . . . . .	87
is_vector . . . . .	87
setopts_are_equal . . . . .	88
setopts_common_elements . . . . .	88
setopts_count_exclusive_to_first . . . . .	89
setopts_exclusive_to_first . . . . .	89
util_count_duplicates . . . . .	90
util_count_missing . . . . .	90
util_get_duplicated_values . . . . .	91

**Index** **92**

---

assert	<i>Assert that conditions are met</i>
--------	---------------------------------------

---

**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 <code>{failed_expressions}</code> to insert a list of the failed expressions. The string can also include <code>{?s}</code> and <code>{?is/are}</code> to insert the correct pluralization for the list of failed expressions.
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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	<i>List assertion names</i>
-----------------	-----------------------------

---

**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	<i>Count tests per Assertion</i>
-----------------	----------------------------------

---

**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\_between     *Assert input is between a specified minimum and maximum value*

---

### Description

Assert all elements in a numeric vector/matrix are between some minimum and maximum values.

### Usage

```
assert_all_between(
  x,
  minimum,
  maximum,
  inclusive = TRUE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)
```

### Arguments

x	An object to check
minimum	The minimum value to compare against (number)
maximum	The maximum value to compare against (number)
inclusive	should comparisons allow equality with the minimum and maximum values? (flag)
msg	A character string containing the error message to display if x is not between the specified minimum and maximum values (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 between the specified minimum and maximum values, otherwise aborts with the error message specified by msg

### Examples

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

```
})
```

---

```
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 <a href="#">cli_abort</a> 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

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

### 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 <a href="#">cli_abort</a> 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\_finite      *Assert that all values are finite*

---

### Description

Check that all values in `x` are finite (not NA, NaN, Inf, or -Inf).

### Usage

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

### Arguments

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

### Value

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

### Examples

```
try({
  assert_all_finite(c(1, 2, 3)) # Passes
  assert_all_finite(c(1, Inf)) # 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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\_all\_less\_than *Assert input is less than a specified maximum value*

---

## Description

Assert all elements in a numeric vector/matrix are below some maximum value.

**Usage**

```

assert_all_less_than(
  x,
  maximum,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)

```

**Arguments**

x	An object to check
maximum	The maximum value to compare against (number)
msg	A character string containing the error message to display if x is not less than the specified maximum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 less than the specified maximum value, otherwise aborts with the error message specified by msg

**Examples**

```

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

```

---

```
assert_all_less_than_or_equal_to
```

*Assert input is less than or equal to a specified maximum value*

---

**Description**

Assert all elements in a numeric vector/matrix are below or equal to some maximum value.

**Usage**

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

**Arguments**

x	An object to check
maximum	The maximum value to compare against
msg	A character string containing the error message to display if x is not less than or equal to the specified maximum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 less than or equal to the specified maximum value, otherwise aborts with the error message specified by msg

**Examples**

```
try({
  assert_less_than_or_equal_to(1, 2) # Passes
  assert_less_than_or_equal_to(c(1, 2, 3), 3) # Passes
  assert_less_than_or_equal_to(3, 2) # Throws error
})
```

---

```
assert_all_strings_contain
```

*Assert all strings contain a regex pattern*

---

**Description**

Assert all elements of a character vector match a regex pattern.

**Usage**

```

assert_all_strings_contain(
  x,
  pattern,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE,
  msg = NULL,
  call = rlang::caller_env(),
  arg_name = NULL
)

```

**Arguments**

x	An object to check
pattern	A regular expression pattern (string)
ignore.case, perl, fixed, useBytes	Logical flags passed to <a href="#">grepl()</a>
msg	A character string containing the error message to display if x does not match pattern
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

**Value**

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

**Examples**

```

try({
  assert_all_strings_contain(c("abc", "a1"), "^a") # Passes
  assert_all_strings_contain(c("abc", "b1"), "^a") # Throws default error
  assert_all_strings_contain(c("abc", "b1"), "^a", msg = "Custom error message") # Throws custom error
})

```

---

 assert\_between

*Assert input is between a specified minimum and maximum value*


---

**Description**

Assert a number is between a specified minimum and maximum value. To check all numbers in a vector / matrix are between minimum and maximum values, see [assert\\_all\\_between\(\)](#)

## Usage

```
assert_between(  
  x,  
  minimum,  
  maximum,  
  inclusive = TRUE,  
  msg = NULL,  
  call = rlang::caller_env(),  
  arg_name = NULL  
)
```

## Arguments

x	An object to check
minimum	The minimum value to compare against
maximum	The maximum value to compare against
inclusive	should comparisons allow equality with the minimum and maximum values? (flag)
msg	A character string containing the error message to display if x is not between the specified minimum and maximum values (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 between the specified minimum and maximum values, otherwise aborts with the error message specified by msg

## Examples

```
try({  
  assert_between(3, 1, 4) # Passes  
  assert_between(3, 1, 4) # Passes  
  assert_between(c(2,3,4), 1, 4) # Throws error (Must be a number)  
  assert_between('A', 1, 4) # Throws error (Must be a number)  
  assert_between(5, 1, 4, msg = "custom error message") # Throws custom error  
})
```

---

assert_character	<i>Assert input is a character vector</i>
------------------	---

---

## 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 <a href="#">cli_abort</a> 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

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

## 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_character_vector(glue::glue('A')) # Throws error  
})
```

---

`assert_character_vector_or_glue`*Assert input is a character vector / glue vector*

---

## Description

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

## Usage

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

## Arguments

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

## Value

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

## Examples

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

---

assert_class	<i>Assert object belongs to class</i>
--------------	---------------------------------------

---

**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 <a href="#">cli_abort</a> 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_connection	<i>Assert input is a database connection</i>
-------------------	--

---

**Description**

Assert the input object is a database connection, specifically of the "DBIConnection" class, which is the standard virtual class used by the DBI package for database connections. Note this assertion does not test if the database connection is valid and/or active.

**Usage**

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

**Arguments**

x	An object to assert is a database connection
msg	A custom error message displayed if x is not a valid database connection.
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
arg_name	Advanced use only. Name of the argument passed (default: NULL, will automatically extract arg_name).

**Details**

This function is designed for use with objects inheriting from the "DBIConnection" class, which is used widely across database connection implementations in R. As other database interface packages are introduced, additional checks may be added to support other connection classes.

**Value**

invisible(TRUE) if x is a valid database connection, otherwise aborts with an error message.

**Examples**

```
try({
  # Assuming a valid DBI connection `conn`:
  assert_connection(conn) # Passes if `conn` is a DBI connection

  assert_connection(42) # Fails with error message
})
```

---

assert_create	<i>Create an assertion function</i>
---------------	-------------------------------------

---

**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 <ol style="list-style-type: none"> <li>1. {arg_name} to refer to the name of the variable supplied to the assertion.</li> <li>2. {arg_value} to refer to the value of the variable supplied to the assertion</li> <li>3. {code_to_evaluate} to evaluate the code within the error message. Replace code_to_evaluate with your code</li> <li>4. {.strong bold_text} to perform inline formatting. Replace bold_text with your text. See <a href="#">cli documentation</a> for details</li> </ol>

**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	<i>Assert input is a data frame</i>
------------------	-------------------------------------

---

**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 <a href="#">cli_abort</a> 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**

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

**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 <a href="#">cli_abort</a> 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	<i>Assert that the input objects are equal</i>
--------------	--

---

### 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

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

### 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	<i>Assert object does not include any illegal values</i>
-----------------	--

---

**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 <a href="#">cli_abort</a> 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

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

### 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

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

## 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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_finite	<i>Assert that a value is finite</i>
---------------	--------------------------------------

---

**Description**

Check that a single numeric value is finite (not NA, NaN, Inf, or -Inf). For vectorized version see [assert\\_all\\_finite\(\)](#).

**Usage**

```
assert_finite(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 <a href="#">cli_abort</a> 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 finite, otherwise aborts with the error message specified by msg

**Examples**

```
try({
  assert_finite(1) # Passes
  assert_finite(Inf) # Throws error
})
```

---

assert_flag	<i>Assert input is a scalar logical</i>
-------------	---

---

**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 <a href="#">cli_abort</a> 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	<i>Assert input is a function</i>
-----------------	-----------------------------------

---

**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 <a href="#">cli_abort</a> 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`*Assert function expects specific parameter names*

---

### Description

Assert that a function signature includes required set of parameter names in its formal argument list, regardless of whether those parameters have default values. The `...` argument is ignored.

### Usage

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

### Arguments

<code>x</code>	a function to check for required parameter names
<code>required</code>	a character vector of parameter names that must appear in the function signature (order does not matter)
<code>msg</code>	The error message thrown if the assertion fails (string)
<code>call</code>	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
<code>arg_name</code>	Advanced use only. Name of the argument passed (default: <code>NULL</code> , will automatically extract <code>arg_name</code> ).

### Value

`invisible(TRUE)` if function `x` declares all required parameters, otherwise aborts with the error message specified by `msg`

### Examples

```
my_fun <- function(x, y = 1, ...) x + y  
assert_function_expects(my_fun, c("x", "y"))  
  
try({  
  assert_function_expects(my_fun, c("x", "z"))  
})
```

---

```
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

<code>x</code>	a function to check has exactly <code>N</code> arguments
<code>n</code>	number of arguments that must be expected by function to pass assertion (integer)
<code>dots</code>	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)
<code>msg</code>	The error message thrown if the assertion fails (string)
<code>call</code>	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
<code>arg_name</code>	Advanced use only. Name of the argument passed (default: NULL, will automatically extract <code>arg_name</code> ).

## Value

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

---

`assert_function_variadic`*Assert function is variadic*

---

## Description

Assert that a function signature declares the `...` argument (variadic arguments).

## Usage

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

## Arguments

<code>x</code>	a function to check includes <code>...</code> in its signature
<code>msg</code>	The error message thrown if the assertion fails (string)
<code>call</code>	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
<code>arg_name</code>	Advanced use only. Name of the argument passed (default: <code>NULL</code> , will automatically extract <code>arg_name</code> ).

## Value

`invisible(TRUE)` if function `x` declares `...`, otherwise aborts with the error message specified by `msg`

## Examples

```
my_fun <- function(x, ...) x  
assert_function_variadic(my_fun)  
  
try({  
  my_fun_no_dots <- function(x) x  
  assert_function_variadic(my_fun_no_dots)  
})
```

---

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 <a href="#">cli_abort</a> 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

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

### 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	<i>Assert that the input object is identical to a specified value</i>
------------------	---

---

**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 <a href="#">cli_abort</a> 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	<i>Assert object includes required</i>
-----------------	--

---

**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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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\_length

*Assert Length*

---

**Description**

Assert object has a specific length

**Usage**

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

**Arguments**

x	object to check length of
length	expected length (number)
msg	custom error message
call	(logical) whether to preserve call in error message
arg_name	(character) name of argument being tested

**Value**

invisible(TRUE)

---

assert\_length\_greater\_than  
*Assert Length Greater Than*

---

**Description**

Assert object length is greater than a threshold

**Usage**

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

**Arguments**

x	object to check length of
length	expected length (number)
msg	custom error message
call	(logical) whether to preserve call in error message
arg_name	(character) name of argument being tested

**Value**

invisible(TRUE)

---

assert\_length\_greater\_than\_or\_equal\_to  
*Assert Length Greater Than or Equal To*

---

**Description**

Assert object length is greater than or equal to a threshold

**Usage**

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

**Arguments**

x	object to check length of
length	expected length (number)
msg	custom error message
call	(logical) whether to preserve call in error message
arg_name	(character) name of argument being tested

**Value**

invisible(TRUE)

---

assert\_length\_less\_than  
*Assert Length Less Than*

---

**Description**

Assert object length is less than a threshold

**Usage**

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

**Arguments**

x	object to check length of
length	expected length (number)
msg	custom error message
call	(logical) whether to preserve call in error message
arg_name	(character) name of argument being tested

**Value**

invisible(TRUE)

---

assert\_length\_less\_than\_or\_equal\_to  
*Assert Length Less Than or Equal To*

---

**Description**

Assert object length is less than or equal to a threshold

**Usage**

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

**Arguments**

x	object to check length of
length	expected length (number)
msg	custom error message
call	(logical) whether to preserve call in error message
arg_name	(character) name of argument being tested

**Value**

invisible(TRUE)

---

assert_less_than	<i>Assert input is less than some maximum value</i>
------------------	---

---

### Description

Assert a number is less than a specified maximum value. To check all numbers in a vector / matrix are below a maximum value, see [assert\\_all\\_less\\_than\(\)](#)

### Usage

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

### Arguments

x	An object to check
maximum	The maximum value to compare against (number)
msg	A character string containing the error message to display if x is not less than the specified maximum value (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 less than the specified maximum value, otherwise aborts with the error message specified by msg

### Examples

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

---

`assert_less_than_or_equal_to`*Assert input is less than or equal to a specified maximum value*

---

### Description

Assert a number is less than or equal to a specified maximum value. For vectorized version see [assert\\_all\\_less\\_than\\_or\\_equal\\_to\(\)](#)

### Usage

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

### Arguments

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

### Value

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

### Examples

```
try({  
  assert_less_than_or_equal_to(1, 2) # Passes  
  assert_less_than_or_equal_to(c(1, 2, 3), 3) # Throws error  
  assert_less_than_or_equal_to(3, 2) # Throws error  
})
```

---

assert_list	<i>Assert input is a list</i>
-------------	-------------------------------

---

## 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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	<i>Assert input is a matrix</i>
---------------	---------------------------------

---

**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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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_non_null	<i>Assert that the input is not NULL</i>
-----------------	--

---

**Description**

This function asserts that the input is not NULL and aborts with an error message if it is.

**Usage**

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

**Arguments**

x	A value to check.
msg	A character string containing the error message to display if x is NULL.
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 not NULL, otherwise aborts with the error message specified by msg.

**Examples**

```
# Passes for non-NULL
assert_non_null(1)

try({
  # Throws default error for NULL
  assert_non_null(NULL)

  # Throws custom error message
  assert_non_null(NULL, msg = "Custom error message")
})
```

---

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 <a href="#">cli_abort</a> 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	<i>Assert that the input vector has no missing values</i>
-------------------	---

---

## 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 <a href="#">cli_abort</a> 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_null	<i>Assert that the input is NULL</i>
-------------	--------------------------------------

---

**Description**

This function asserts that the input is NULL and aborts with an error message if it is not.

**Usage**

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

**Arguments**

x	A value to check.
msg	A character string containing the error message to display if x is not NULL.
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 NULL, otherwise aborts with the error message specified by msg.

**Examples**

```
assert_null(NULL) # Passes

try({
  assert_null(1) # Throws default error
  assert_null(1, msg = "Custom error message") # Throws custom error
})
```

---

assert_number	<i>Assert input is a number</i>
---------------	---------------------------------

---

**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 <a href="#">cli_abort</a> 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	<i>Assert input is numeric</i>
----------------	--------------------------------

---

**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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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\_one\_of                      *Check if a scalar value is one of the acceptable values*

---

### Description

Assert x is one of the values of y.

### Usage

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

### Arguments

x	A scalar value to check
y	A vector of acceptable 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 <a href="#">cli_abort</a> 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 scalar and is one of the values in y, otherwise throws an error

### Examples

```
assert_one_of(3, 1:5) # Passes because 3 is in 1:5
assert_one_of("A", c("A", "B", "C")) # Passes because "A" is in the vector

try({
  assert_one_of("D", c("A", "B", "C")) # Throws error because "D" is not in the vector
})
```

---

assert\_packages\_installed  
*Assert packages are installed*

---

### Description

Assert that one or more packages are installed.

**Usage**

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

**Arguments**

x	Package name(s) (character vector)
msg	A character string containing the error message to display if the package is not installed
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 installed, otherwise aborts with the message specified by msg

**Examples**

```
try({
  assert_packages_installed("stats") # Passes
  assert_packages_installed(c("stats", "utils")) # Passes
  assert_packages_installed("notapackage123") # Throws Error
})
```

---

assert_reactive	<i>Assert that x is reactive</i>
-----------------	----------------------------------

---

**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 <a href="#">cli_abort</a> 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_scalar	<i>Assert input is a scalar</i>
---------------	---------------------------------

---

**Description**

Assert that an object is a scalar, meaning it is a length 1 atomic vector (such as `numeric(1)`, `character(1)` or `logical(1)`). Note lists, data.frames and matrices are never considered scalar objects, even if they have only one element.

**Usage**

```
assert_scalar(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
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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, otherwise aborts with the error message specified by msg

**Examples**

```
# Pass when value is scalar
assert_scalar(5) # Passes
assert_scalar("single string") # Passes
assert_scalar(TRUE) # Passes

# Fail when value is not
try({
  assert_scalar(c(1, 2, 3)) # Throws default error
  assert_scalar(matrix(1:4, 2, 2)) # Throws default error
})
```

---

assert_set_equal	<i>Check if two sets are identical</i>
------------------	--

---

**Description**

This function checks that x and y contain exactly the same elements, ignoring order and duplicates.

**Usage**

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

**Arguments**

x	A vector to compare
y	Another vector to compare with x
msg	The error message thrown if the assertion fails (string)
call	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> 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 and y contain all the same elements (ignoring order and duplicates), otherwise throws an error.

**Examples**

```
# Passes because elements are the same, order doesn't matter
assert_set_equal(c(1, 2, 3), c(3, 2, 1))

# Passes because elements are identical
assert_set_equal(c("A", "B", "C"), c("C", "A", "B"))

try({
  # Throws error because elements are not identical
  assert_set_equal(c(1, 2, 3), c(1, 2))

  # Throws error because elements differ
  assert_set_equal(c("A", "B"), c("A", "B", "C"))
})
```

---

assert_string	<i>Assert input is a character string</i>
---------------	---

---

**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 <a href="#">cli_abort</a> 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_string_contains`*Assert string contains a regex pattern*

---

## Description

Assert a string matches a regex pattern.

## Usage

```
assert_string_contains(  
  x,  
  pattern,  
  ignore.case = FALSE,  
  perl = FALSE,  
  fixed = FALSE,  
  useBytes = FALSE,  
  msg = NULL,  
  call = rlang::caller_env(),  
  arg_name = NULL  
)
```

## Arguments

<code>x</code>	An object to check
<code>pattern</code>	A regular expression pattern (string)
<code>ignore.case</code> , <code>perl</code> , <code>fixed</code> , <code>useBytes</code>	Logical flags passed to <code>grepl()</code>
<code>msg</code>	A character string containing the error message to display if <code>x</code> does not match <code>pattern</code>
<code>call</code>	Only relevant when pooling assertions into multi-assertion helper functions. See <a href="#">cli_abort</a> for details.
<code>arg_name</code>	Advanced use only. Name of the argument passed (default: <code>NULL</code> , will automatically extract <code>arg_name</code> ).

## Value

`invisible(TRUE)` if `x` matches `pattern`, otherwise aborts with the error message specified by `msg`

## Examples

```
try({  
  assert_string_contains("abc", "^a") # Passes  
  assert_string_contains("abc", "^b") # Throws default error  
  assert_string_contains("abc", "^b", msg = "Custom error message") # Throws custom error  
})
```

---

assert_subset	<i>Check if a vector is a subset of another</i>
---------------	---

---

**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**

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

**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	<i>Assert input is a vector</i>
---------------	---------------------------------

---

**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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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 <a href="#">cli_abort</a> 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

---

 contains\_pattern *Check if strings contain a regex pattern*


---

**Description**

This helper checks whether all elements of a character vector match a regex pattern.

**Usage**

```
contains_pattern(
  x,
  pattern,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

**Arguments**

x	A character vector to check
pattern	A regular expression pattern (string)
ignore.case, perl, fixed, useBytes	Logical flags passed to <a href="#">grepl()</a>

**Value**

TRUE if all strings in `x` match `pattern`, otherwise FALSE or a string error message.

---

<code>excludes_advanced</code>	<i>Check if an object does not contain prohibited elements</i>
--------------------------------	--

---

**Description**

This function checks that `x` does not include any of the `illegal` elements. `x` must be the same type as `illegal`. Factors are treated as character vectors.

**Usage**

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

**Arguments**

<code>x</code>	An object to check
<code>illegal</code>	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

---

<code>format_as_bullets</code>	<i>Preprocess character vectors for <code>cli::cli_abort()</code></i>
--------------------------------	---

---

**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**

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

**Value**

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

---

format_inline	<i>Preprocess character vectors for cli package functions</i>
---------------	---

---

**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	<i>Check if a named object has all specified names</i>
---------------	--

---

**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	<i>Has Extension</i>
---------------	----------------------

---

**Description**

Has Extension

**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	<i>Check if a vector has missing values</i>
--------------------	---

---

**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	<i>Check if a vector has no duplicates</i>
-------------------	--

---

**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	<i>Check if an object contains required elements</i>
-------------------	--

---

**Description**

This function checks that `x` includes all of the required elements. `x` must be the same type as required. Factors are treated as character vectors.

**Usage**

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

**Arguments**

<code>x</code>	An object to check
<code>required</code>	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_between	<i>Check if a numeric vector is between a specified minimum and maximum value</i>
------------	---

---

**Description**

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

**Usage**

```
is_between(x, minimum, maximum, inclusive = TRUE)
```

**Arguments**

<code>x</code>	a numeric vector to check
<code>minimum</code>	The minimum value to compare against
<code>maximum</code>	The maximum value to compare against
<code>inclusive</code>	should comparisons allow equality with the minimum and maximum values? (flag)

**Value**

A logical value indicating whether all elements of the numeric vector `x` are between the specified minimum and maximum values

**Examples**

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

---

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\_character\_vector\_or\_glue  
*Check if an object is a character vector*

---

**Description**

Differs from [is\\_character\\_vector\(\)](#) in that it permits glue character vectors to pass.

**Usage**

```
is_character_vector_or_glue(x)
```

**Arguments**

`x`                    An object to check.

**Value**

A logical value indicating whether x is a character vector or glue vector.

---

is_equal	<i>Check equality of two objects</i>
----------	--------------------------------------

---

**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	<i>Check if a value is a logical flag</i>
---------	---

---

**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	<i>Check if x is a flag</i>
------------------	-----------------------------

---

**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	<i>Check if a numeric vector is greater than a specified minimum value</i>
-----------------	--

---

**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	<i>Check if a numeric vector is greater than or equal to a specified minimum value</i>
-----------------------------	--

---

**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	<i>Check if two objects are identical</i>
--------------	---

---

**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_less_than	<i>Check if a numeric vector is less than a specified maximum value</i>
--------------	---

---

**Description**

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

**Usage**

```
is_less_than(x, maximum)
```

**Arguments**

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

**Value**

A logical value indicating whether all elements of the numeric vector x are less than the specified maximum value

**Examples**

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

---

is_less_than_or_equal_to	<i>Check if a numeric vector is less than or equal to a specified maximum value</i>
--------------------------	---

---

**Description**

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

**Usage**

```
is_less_than_or_equal_to(x, maximum)
```

**Arguments**

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

**Value**

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

**Examples**

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

---

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	<i>Check if an object is a single number</i>
-----------	--

---

**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	<i>Check if x is a number</i>
--------------------	-------------------------------

---

**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	<i>Check equality of type</i>
--------------	-------------------------------

---

**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	<i>Check if an object is a single string</i>
-----------	--

---

**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	<i>Check if x is a string</i>
--------------------	-------------------------------

---

**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	<i>Check if one set is a subset of another</i>
-----------	--

---

**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	<i>Check if one set is a superset of another</i>
-------------	--

---

**Description**

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

**Usage**

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

**Arguments**

<i>x</i>	A numeric, character, or logical vector.
<i>y</i>	A numeric, character, or logical vector.

**Value**

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

---

is_vector	<i>Check if an object is a vector This function checks if an object is a vector</i>
-----------	---

---

**Description**

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

**Usage**

```
is_vector(x)
```

**Arguments**

<i>x</i>	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_exclusive_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_exclusive_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_exclusive_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_exclusive_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

- \* **advanced**
    - is\_flag\_advanced, 77
    - is\_non\_empty\_string\_advanced, 82
    - is\_number\_advanced, 83
    - is\_string\_advanced, 86
  - \* **assert\_comparison**
    - assert\_all\_between, 6
    - assert\_all\_greater\_than, 10
    - assert\_all\_less\_than, 12
    - assert\_between, 15
    - assert\_greater\_than, 37
    - assert\_less\_than, 45
  - \* **assert\_create**
    - assert, 4
    - assert\_create, 21
  - \* **assert\_file**
    - assert\_all\_directories\_exist, 7
    - assert\_all\_files\_exist, 8
    - assert\_all\_files\_have\_extension, 9
    - assert\_directory\_does\_not\_exist, 24
    - assert\_directory\_exists, 25
    - assert\_file\_does\_not\_exist, 29
    - assert\_file\_exists, 30
    - assert\_file\_has\_extension, 30
  - \* **assert\_has**
    - assert\_class, 20
    - assert\_no\_duplicates, 53
    - assert\_no\_missing, 54
  - \* **assert\_includes**
    - assert\_excludes, 27
    - assert\_includes, 39
  - \* **assert\_numerical**
    - assert\_all\_finite, 10
    - assert\_finite, 31
    - assert\_whole\_number, 65
  - \* **assert\_package**
    - assert\_packages\_installed, 58
  - \* **assert\_regex**
    - assert\_all\_strings\_contain, 14
    - assert\_string\_contains, 63
    - contains\_pattern, 67
  - \* **assert\_type**
    - assert\_character, 17
    - assert\_character\_vector, 18
    - assert\_character\_vector\_or\_glue, 19
    - assert\_connection, 20
    - assert\_dataframe, 23
    - assert\_factor\_vector, 28
    - assert\_flag, 32
    - assert\_function, 33
    - assert\_int, 40
    - assert\_list, 47
    - assert\_logical, 48
    - assert\_logical\_vector, 48
    - assert\_matrix, 49
    - assert\_non\_empty\_string, 51
    - assert\_number, 55
    - assert\_numeric, 56
    - assert\_numeric\_vector, 57
    - assert\_reactive, 59
    - assert\_scalar, 60
    - assert\_string, 62
    - assert\_vector, 64
  - \* **is\_type**
    - has\_class, 70
    - is\_character\_vector, 75
    - is\_character\_vector\_or\_glue, 75
    - is\_flag, 77
    - is\_list, 81
    - is\_logical\_vector, 82
    - is\_number, 83
    - is\_numeric\_vector, 84
    - is\_reactive, 84
    - is\_string, 85
    - is\_vector, 87
- all.equal(), 26, 76

assert, 4  
assert\_all\_between, 6  
assert\_all\_between(), 15  
assert\_all\_directories\_exist, 7  
assert\_all\_directories\_exist(), 25  
assert\_all\_files\_exist, 8  
assert\_all\_files\_exist(), 30  
assert\_all\_files\_have\_extension, 9  
assert\_all\_finite, 10  
assert\_all\_finite(), 31  
assert\_all\_greater\_than, 10  
assert\_all\_greater\_than(), 37  
assert\_all\_greater\_than\_or\_equal\_to, 11  
assert\_all\_greater\_than\_or\_equal\_to(), 38  
assert\_all\_less\_than, 12  
assert\_all\_less\_than(), 45  
assert\_all\_less\_than\_or\_equal\_to, 13  
assert\_all\_less\_than\_or\_equal\_to(), 46  
assert\_all\_strings\_contain, 14  
assert\_between, 15  
assert\_character, 17  
assert\_character\_vector, 18  
assert\_character\_vector(), 17  
assert\_character\_vector\_or\_glue, 19  
assert\_class, 20  
assert\_connection, 20  
assert\_create, 21  
assert\_create(), 21  
assert\_create\_chain, 22  
assert\_create\_chain(), 21  
assert\_dataframe, 23  
assert\_directory\_does\_not\_exist, 24  
assert\_directory\_exists, 25  
assert\_directory\_exists(), 7  
assert\_equal, 26  
assert\_excludes, 27  
assert\_factor\_vector, 28  
assert\_file\_does\_not\_exist, 29  
assert\_file\_does\_not\_exist(), 24  
assert\_file\_exists, 30  
assert\_file\_exists(), 8  
assert\_file\_has\_extension, 30  
assert\_finite, 31  
assert\_flag, 32  
assert\_function, 33  
assert\_function\_expects, 34  
assert\_function\_expects\_n\_arguments, 35  
assert\_function\_variadic, 36  
assert\_greater\_than, 37  
assert\_greater\_than\_or\_equal\_to, 38  
assert\_identical, 39  
assert\_includes, 39  
assert\_int, 40  
assert\_length, 41  
assert\_length\_greater\_than, 42  
assert\_length\_greater\_than\_or\_equal\_to, 43  
assert\_length\_less\_than, 43  
assert\_length\_less\_than\_or\_equal\_to, 44  
assert\_less\_than, 45  
assert\_less\_than\_or\_equal\_to, 46  
assert\_list, 47  
assert\_logical, 48  
assert\_logical\_vector, 48  
assert\_logical\_vector(), 48  
assert\_matrix, 49  
assert\_names\_include, 50  
assert\_no\_duplicates, 53  
assert\_no\_missing, 54  
assert\_non\_empty\_string, 51  
assert\_non\_null, 52  
assert\_null, 55  
assert\_number, 55  
assert\_numeric, 56  
assert\_numeric\_vector, 57  
assert\_numeric\_vector(), 56  
assert\_one\_of, 58  
assert\_packages\_installed, 58  
assert\_reactive, 59  
assert\_scalar, 60  
assert\_set\_equal, 61  
assert\_string, 62  
assert\_string\_contains, 63  
assert\_subset, 64  
assert\_vector, 64  
assert\_whole\_number, 65  
assertion\_names, 5  
assertion\_tests, 5  
check\_all\_assertions\_are\_tested\_enough, 66  
cli\_abort, 4, 6–21, 23–41, 45–67  
common\_roxygen\_params, 67

contains\_pattern, 67

excludes\_advanced, 68

format\_as\_bullets, 68

format\_inline, 69

grepl(), 15, 63, 67

has\_all\_names, 69

has\_class, 70

has\_duplicates, 70

has\_extension, 71

has\_missing\_values, 71

has\_no\_duplicates, 72

has\_no\_missing\_values, 73

includes, 73

includes\_advanced, 74

is\_between, 74

is\_character\_vector, 75

is\_character\_vector(), 75

is\_character\_vector\_or\_glue, 75

is\_equal, 76

is\_flag, 77

is\_flag\_advanced, 77

is\_greater\_than, 78

is\_greater\_than\_or\_equal\_to, 78

is\_identical, 79

is\_less\_than, 80

is\_less\_than\_or\_equal\_to, 80

is\_list, 81

is\_logical\_vector, 82

is\_non\_empty\_string\_advanced, 82

is\_number, 83

is\_number\_advanced, 83

is\_numeric\_vector, 84

is\_reactive, 84

is\_same\_type, 85

is\_string, 85

is\_string\_advanced, 86

is\_subset, 86

is\_superset, 87

is\_vector, 87

setopts\_are\_equal, 88

setopts\_common\_elements, 88

setopts\_count\_exclusive\_to\_first, 89

setopts\_exclusive\_to\_first, 89

util\_count\_duplicates, 90

util\_count\_missing, 90

util\_get\_duplicated\_values, 91