Package 'favr'

August 20, 2026

Title Function Argument Validation

Version 2.0.0

Description Validate function arguments succinctly with informative error messages.

License MIT + file LICENSE

Encoding UTF-8

Imports cli, lifecycle, methods, rlang (>= 1.1.0), tidyselect, vctrs

Suggests testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL https://lj-jenkins.github.io/favr/,
https://github.com/LJ-Jenkins/favr

Depends R (>= 4.1.0)

BugReports https://github.com/LJ-Jenkins/favr/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Luke Jenkins [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-7206-7242>)

Maintainer Luke Jenkins <luke-jenkins-dev@outlook.com>

Repository CRAN

Date/Publication 2026-08-20 14:10:28 UTC

Contents

`abortifnot` *Ensure the truth of R expressions*

Description

If any of the expressions in `...` are not (`all`) `TRUE`, `cli_abort()` is called, producing an error message indicating the first expression which was not (`all`) `TRUE`.

For `abortif()`, the opposite is true, i.e. expressions should evaluate to (`all`) `FALSE` for no error to occur.

Usage

```
abortifnot(
  ...,
  message = NULL,
  call = .envir,
  .envir = parent.frame(),
  .frame = .envir,
  abort_args = NULL
)

abortif(
  ...,
  message = NULL,
  call = .envir,
  .envir = parent.frame(),
  .frame = .envir,
  abort_args = NULL
)
```

Arguments

`...`	Any number of R expressions, which should each evaluate to (a `logical` vector of **all**) `TRUE` for no error to occur (`FALSE` for `abortif()`). Non-`logical` and `NA` values will trigger an error. If an expression is named, the name will be used in the error message instead of the default message or the `message` argument.

message	Default error message for non-named expressions.
call	An execution environment, defused function call, or NULL. Passed to cli_abort().
.envir	Environment to evaluate the cli formatting of the error message in. Passed to cli_abort().
.frame	The throwing context. Passed to cli_abort().
abort_args	A list of additional arguments to pass to abort() (forwarded from cli_abort()).

Value

NULL invisibly if the checks pass, otherwise an error is thrown.

See Also

stopifnot() for the base **R** function this is based on.

check() and check_with() for a non data-masked and data-masked version of abortifnot() with tidy evaluation and injection support.

Examples

```
abortifnot(1 == 1, all.equal(pi, 3.14159265), 1 < 2) # all TRUE

m <- matrix(c(1, 3, 3, 1), 2, 2)
abortifnot(m == t(m), diag(m) == rep(1, 2)) # all TRUE

abortifnot(1) |> try()

# A custom error message can be given for each expression:
m[1, 2] <- 12
abortifnot("{.var m} must be {.cls symmetric}" = m == t(m)) |>
  try()

# Alternatively, one error message can be used for all
# expressions.
abortifnot(
  m[1, 1] == 1,
  diag(m) == rep(2, 2),
  message = "{.var m} has a diagonal of: {diag(m)}"
) |> try()

# The `call` argument can be used to specify where the
# error occurs, by default this is the caller environment.
myfunc <- function(x) abortifnot(x)
myfunc(FALSE) |> try()

# abortif() errors if any argument does not evaluate to
# (all) FALSE.
abortif(c(T, F)) |> try()
abortif(c(T, NA)) |> try()
```

array-type-checks *Array type checks*

Description

Check if inputs are expected types and throw an error if not.

Usage

```
check_array(
  x,
  n = NULL,
  nrow = NULL,
  ncol = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_matrix(
  x,
  n = NULL,
  nrow = NULL,
  ncol = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_table(
  x,
  n = NULL,
  nrow = NULL,
  ncol = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.

n, nrow, ncol	The expected length, number of columns, or number of rows of x.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
finite	Whether x is required to contain only finite values (i.e. no NA, Inf, -Inf, or NaN).
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Details

These functions can be used with the bare() modifier to check if an object is a bare R object (i.e. has no class attribute), and the length modifiers at_least(), at_most(), and in_range() to modify the behaviour of the length checking n, nrow, and ncol arguments.

Note that the bare() modifier uses is.object() for check_array() and check_matrix(), but uses the S3-style check for check_table(), which checks if "table" is the first class in the class vector.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

These check functions are wrappers of their corresponding base functions is.array(), is.matrix() and is.table().

See Also

Other checks: forbidden-value-checks, inheritance-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
a <- array(1:12, dim = c(3, 4))
check_array(a)
check_array(1:12) |> try()

m <- matrix(1:12, nrow = 3)
check_matrix(m)
check_matrix(1:12) |> try()

t <- table(c("a", "b", "a"))
check_table(t)
check_table(1:12) |> try()
```

```
class(m) <- c("my_matrix", class(m))
check_matrix(bare(m)) |> try()

check_array(a, n = 10) |> try()
check_array(a, n = at_least(10))

check_matrix(m, ncol = at_most(3)) |> try()
check_matrix(m, nrow = in_range(1, 10))
```

---

check *Check the truth of tidy evaluated expressions*

Description

If any of the expressions in `...` are not (`all`) `TRUE`, `cli_abort()` is called, producing an error message indicating the first expression which was not (`all`) `TRUE`.

`check_with()` is a data-masked version of `check()`, evaluating the expression in the context of `.data`.

Usage

```
check(
  ...,
  message = NULL,
  call = .envir,
  .envir = parent.frame(),
  .frame = .envir,
  abort_args = NULL
)

check_with(
  .data,
  ...,
  message = NULL,
  call = .envir,
  .envir = parent.frame(),
  .frame = .envir,
  abort_args = NULL
)
```

Arguments

`...`	Any number of R expressions, which should each evaluate to (a `logical` vector of **all**) `TRUE` for no error to occur. Non-`logical` and `NA` values will trigger an error. If an expression is named, the name will be used in the error message instead of the default message or the `message` argument.

message	Default error message for non-named expressions.
call	An execution environment, defused function call, or NULL. Passed to cli_abort().
.envir	Environment to evaluate the cli formatting of the error message in. Passed to cli_abort(). For check_with(), the messages are evaluated in the context of .data and .envir. See examples.
.frame	The throwing context. Passed to cli_abort().
abort_args	A list of additional arguments to pass to abort() (forwarded from cli_abort()).
.data	A data frame, list, or environment to evaluate the expressions in as a data mask.

Value

NULL, called for side effects only.

See Also

abortifnot() for a more performant version without tidy evaluation and injection support.

Examples

```
check(1 == 1, all.equal(pi, 3.14159265), 1 < 2) # all TRUE

data <- data.frame(x = 1:5, y = 6:10)
check_with(data, x < y, is.numeric(x), length(y) < 10) # all TRUE

# A custom error message can be given for each
# expression, with cli formatting.
check(
  "message {.arg 1}" = TRUE, "message {.arg 2}" = FALSE
) |> try()

# check_with() names are also are evaluated in
# the context of `.data` then `.envir`.
x <- "env 'x'"
y <- "env 'y'"
data <- list(x = "data 'x'")
check_with(data, "{x}" = is.numeric(x)) |>
  try()
check_with(data, "{y}" = is.numeric(x)) |>
  try()

# Pronouns are supported in check_with() error
# messages, but must be spaced according to cli
# rules (e.g., use `{ .env$x}` instead of `{.env$x}`).
check_with(data, "{ .env$x}" = is.numeric(x)) |>
  try()

# Alternatively, one error message can be used for all
# expressions.
x <- 1:3
```

```
check(
  x > 0, x < 3,
  message = "{.arg x} has incorrect values: {.val {x}}."
) |> try()

data <- data.frame(x = c("a", "b", "c"))
check_with(data,
  is.numeric(x),
  message = "{.arg x} is not numeric: {.val {x}}."
) |>
  try()

# The `call` argument can be used to specify where the
# error occurs, by default this is the caller environment.
myfunc <- function(x) check(x)
myfunc(FALSE) |> try()

myfunc_with <- function(x, ...) check_with(x, ...)
myfunc_with(list(x = 1), x < 0) |> try()

# check() and check_with() error if any argument does
# not evaluate to (all) FALSE.
check(c(T, F)) |> try()
check_with(list(x = c(T, NA)), x) |>
  try()
```

---

forbidden-value-checks

Forbidden value checks

Description

Check if inputs contain forbidden values and error if so.

Usage

```
check_no_na(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_finite(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
```

```
  call = caller_env()
)

check_unique(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_nzchar(
  x,
  ...,
  allow_all_ws = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
allow_all_ws	Whether x is allowed to contain elements that are all whitespace.

Details

NA checks are done with anyNA();

finite checks are done with any() and is.finite();

unique checks are done with anyDuplicated();

zero chr checks are done with any() and nzchar();

If allow_all_ws = FALSE then whitespace elements are identified using grepl("\\s+", x).

Input types are not checked, they are passed 'as is' to the functions that do the forbidden value checking. The only exception is for NULL inputs, which error if allow_null = FALSE.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

NA_character_ is not considered zero chr nor all whitespace.

See Also

Other checks: array-type-checks, inheritance-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
x <- c(1, 2, NA)
check_no_na(x) |> try()

x <- c(1, 2, Inf)
check_finite(x) |> try()

x <- c(1, 2, 3, 1)
check_unique(x) |> try()

x <- c("a", "b", "")
check_nzchar(x) |> try()

x <- c("a", "b", " ")
check_nzchar(x, allow_all_ws = FALSE) |> try()
```

---

inheritance-checks *Check class inheritance of an object*

Description

Check that an object inherits from a specific class (or classes) and throw an error if not.

Usage

```
check_inherits(
  x,
  class,
  match = c("any", "exact", "all"),
  ...,
  arg = caller_arg(x),
  call = caller_env()
)

check_class(x, class, ..., arg = caller_arg(x), call = caller_env())
```

Arguments

x	An object to check.
class	Character vector of class names to check against.
match	The behaviour to use for inheritance checking. See Details.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Details

The match argument specifies how to check the inheritance:

- "any": the class vector of x must have at least one element in common with class.
- "exact": the class vector of x must be identical to class.
- "all": the class vector of x must contain all elements of class in the supplied order.

check_class() is a utility wrapper around check_inherits() with match = "exact".

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

These check functions are wrappers of their corresponding rlang counterparts.

See Also

Other checks: array-type-checks, forbidden-value-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
# Default behaviour is to check for any inheritance.
x <- structure(1, class = c("a", "b", "c"))
check_inherits(x, c("x", "b", "y"))
check_inherits(x, c("x", "y")) |> try()

# `match = "exact"` checks for exact match of the class vector.
x <- structure(1, class = c("a", "b", "c"))
check_inherits(x, c("a", "b", "c"), match = "exact")
check_inherits(x, c("a", "b")) |> try()

# check_class() is a utility wrapper with match = "exact".
check_class(x, c("a", "b", "c"))
```

```
check_class(x, c("a", "b")) |> try()

# `match = "all"` checks that inheritance is from all
# of the classes in the supplied order.
x <- structure(1, class = c("a", "b", "c", "d", "e"))
check_inherits(x, c("b", "d"), match = "all")
check_inherits(x, c("d", "b"), match = "all") |> try()
```

`modifiers` *Modify the behaviour of type checking functions*

Description

Modify the type-checking, or length-checking behaviour of favr type checking functions.

Usage

```
bare(x, arg = caller_arg(x))

at_least(n, arg = caller_arg(n))

at_most(n, arg = caller_arg(n))

in_range(
  n_min,
  n_max,
  arg_min = caller_arg(n_min),
  arg_max = caller_arg(n_max)
)
```

Arguments

`x`	An object to modify the check behaviour for.
`arg`, `arg_min`, `arg_max`	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
`n`, `n_min`, `n_max`	Single numeric value that is castable to an integer. Must be zero or positive.

Details

Use `bare()` to check if a given object is a bare R object (no class attribute, see `is.object()`), throwing an error if it is not and passing the object on to the check if it is.

For `S3` type checks, `bare()` checks that the object has the expected `S3` type as the **first** element of the class vector.

To modify the behaviour of length checking arguments `n`, `nrow`, and `ncol` (example described for `n`):

- `at_least(n)` means the object must be at least length (`>=`) `n`.
- `at_most(n)` means the object must be at most length (`<=`) `n`.
- `in_range(n_min, n_max)` means the object length must be within the range of (`>=`) `n_min` and (`<=`) `n_max`.

Value

A list of class `favr_modifier` with named elements `obj`, `bare` and `arg` for `bare()`, and `at_least` and/or `at_most` for the length modifiers.

See Also

type-checks, scalar-type-checks, s3-type-checks, property-checks and s3-check-builders for the functions that these modifiers can be used with.

Examples

```
bare(1)
at_least(1)
at_most(1)
in_range(1, 2)

at_least(1.5) |> try()

check_integer(bare(factor(1))) |> try()
check_integer(1:5, n = at_least(10)) |> try()
check_integer(1:5, n = at_most(3)) |> try()
check_integer(1:5, n = in_range(2, 4)) |> try()

x <- as.Date("2000-01-01")
class(x) <- c("my_date", class(x))
check_date(bare(x)) |> try()
```

---

oop-checks *Check if an object is of a specific object-oriented programming type*

Description

Check that an object is an `S3`, `S4`, `S7`, or `R6` object.

Usage

```
check_s3(x, ..., allow_null = FALSE, arg = caller_arg(x), call = caller_env())

check_s4(x, ..., allow_null = FALSE, arg = caller_arg(x), call = caller_env())

check_s7(x, ..., allow_null = FALSE, arg = caller_arg(x), call = caller_env())

check_r6(x, ..., allow_null = FALSE, arg = caller_arg(x), call = caller_env())
```

Arguments

x	An object to check.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Details

S3 checks are performed by checking if the object has a class attribute with is.object() and is not an S4 object.

S4 checks are performed using isS4().

S7 and R6 checks are performed by checking for the inheritance of the S7_object and R6 classes, respectively.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
check_s3(factor("a"))
check_s3(1:3) |> try()

methods::setClass("Person",
  slots = c(name = "character", age = "numeric")
)
x <- methods::new("Person", name = "John", age = 30)

check_s4(x)
check_s4(factor("a")) |> try()

# trivial examples of inheritance checks for S7 and R6 objects
x <- structure(list(), class = "S7_object")
check_s7(x)
check_s7(factor("a")) |> try()

x <- structure(list(), class = "R6")
```

```
check_r6(x)
check_r6(factor("a")) |> try()
```

---

path-checks *File and directory existence checks*

Description

Check if inputs are existing directories or files and throw an error if not.

Usage

```
check_dir(x, ..., arg = caller_arg(x), call = caller_env())

check_file(
  x,
  ...,
  ext = NULL,
  case = TRUE,
  x_arg = caller_arg(x),
  ext_arg = caller_arg(ext),
  call = caller_env()
)

check_ext(
  x,
  ext,
  ...,
  case = TRUE,
  x_arg = caller_arg(x),
  ext_arg = caller_arg(ext),
  call = caller_env()
)
```

Arguments

x	A path to check.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
arg, x_arg, ext_arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of cli_abort() for more information.
ext	A character vector of file extensions to check for.
case	A logical value indicating if the extension check should be case-sensitive. If FALSE, the check will be case-insensitive.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

The checking of extensions is done simply using endsWith().

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
x <- file.path(R.home(), "library", "stats")

check_dir(x)
check_file(x) |> try()

x <- file.path(x, "DESCRIPTION")

check_file(x)
check_dir(x) |> try()
check_file(x, ext = c(".csv", ".xlsx")) |> try()
```

---

property-checks *Object property checks*

Description

Check if inputs have certain properties and error if not.

Usage

```
check_length(
  x,
  n,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_nrow(
  x,
  nrow,
  ...,
```

```
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_ncol(
  x,
  ncol,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_size(
  x,
  n,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_non_empty(x, ..., arg = caller_arg(x), call = caller_env())

check_named(
  x,
  ...,
  unique = FALSE,
  allow_empty = TRUE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
n, nrow, ncol	The expected length/size, number of columns, or number of rows of x.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
unique	Whether x is required to have unique names.

allow_empty Whether x is allowed to have empty names ("").

Details

check_size() uses vec_size() to determine the size of x, as opposed to length() which is used by check_length().

Input types are not checked, they are passed 'as is' to the functions that do the property checking. The only exception is for NULL inputs, which error if allow_null = FALSE.

check_length(), check_size(), check_nrow() and check_ncol() can be used with the length modifiers at_least(), at_most(), and in_range() to modify the behaviour of the length checking n, nrow, or ncol arguments.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, path-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
x <- c(1, 2, NA)
check_length(x, 4) |> try()
check_size(x, 4) |> try()

# length modifiers can be used
check_length(x, at_most(2)) |> try()

x <- data.frame(x = 1)
check_nrow(x, 2) |> try()
check_ncol(x, in_range(2, 4)) |> try()
check_size(x, at_least(2)) |> try()

x <- numeric(0)
check_non_empty(x) |> try()
check_non_empty(NULL) |> try()

x <- c(1, 2, 3)
check_named(x) |> try()
names(x) <- c("a", "b", "a")
check_named(x, unique = TRUE) |> try()
names(x) <- c("a", "b", "")
check_named(x, allow_empty = FALSE) |> try()

check_length(NULL, 2, allow_null = TRUE)
```

s3-check-builders *S3 check builders*

Description

Check builders for S3 types. These functions can be used to create custom S3 type checks in the style of favr.

Usage

```
s3_vec_check(
  x,
  n,
  type,
  type_msg = paste0("a {.cls ", type, "}"),
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

s3_df_check(
  x,
  nrow,
  ncol,
  type,
  type_msg = paste0("a {.cls ", type, "}"),
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
n, nrow, ncol	The expected length, number of columns, or number of rows of x.
type	The expected S3 type of x.
type_msg	A message describing the expected S3 type of x, for use in error messages not relating to inheritance, optionally with cli formatting. See details.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.

call — The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Details

Inputs are passed to check_inherits() to check that x inherits from the expected S3 type. This means that error messages about inheritance will always show the expected S3 type in the cli format of {.cls <expected_s3_type>}.

The type_msg argument is used to customise the error message when a different check fails (e.g., length), where the grammar may require different phrasing. For example, the default value is "a {.cls <expected_s3_type>}", but many favr functions use "a {.cls <expected_s3_type>} vector". Also consider where 'an' is more appropriate than 'a'.

These functions can be used with the bare() modifier to check if an object is a bare S3 object (where the expected S3 type is the first class in the class attribute of x), and the length modifiers at_least(), at_most(), and in_range() to modify the behaviour of the length checking n, nrow, and ncol arguments.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

Although named _vec and _df, these functions could be used to check any S3 type, not just vectors and data frames. Their names are intended to indicate the expected behaviour of the check - for types that would use either the length or dimension checking arguments.

Examples

```
# Create a custom type check for a hypothetical "my_class" S3 class
check_my_class <- function(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
) {
  s3_vec_check(
    x,
    n,
    type = "my_class",
    type_msg = "a {.cls my_class} vector",
    ...,
    allow_null = allow_null,
    arg = arg,
    call = call
  )
}
```

```
# inheritance errors use 'type'
check_my_class(1L) |> try()

x <- structure(1:3, class = "my_class")
check_my_class(x)
check_my_class(NULL, allow_null = TRUE)

# other errors use 'type_msg'
check_my_class(x, n = 2) |> try()
check_my_class(x, n = at_least(4)) |> try()
check_my_class(x, n = at_most(2)) |> try()
check_my_class(x, n = in_range(1, 2)) |> try()

class(x) <- c("another_class", class(x))
check_my_class(bare(x)) |> try()
```

---

s3-type-checks *S3 type checks*

Description

Check if inputs are expected S3 types and throw an error if not.

Usage

```
check_date(
  x,
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_posixct(
  x,
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_posixlt(
  x,
```

```
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_factor(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_ordered(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_vctr(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_list_of(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_data_frame(
  x,
```

```
  nrow = NULL,
  ncol = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_tibble(
  x,
  nrow = NULL,
  ncol = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_data_table(
  x,
  nrow = NULL,
  ncol = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_tidytable(
  x,
  nrow = NULL,
  ncol = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
n, nrow, ncol	The expected length, number of columns, or number of rows of x.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_na	Whether x is allowed to contain NA values.
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.

call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
finite	Whether x is required to contain only finite values (i.e. no NA, Inf, -Inf, or NaN).

Details

These functions can be used with the bare() modifier to check if an object is a bare S3 object (where the expected S3 type is the first class in the class attribute of x), and the length modifiers at_least(), at_most(), and in_range() to modify the behaviour of the length checking n, nrow, and ncol arguments.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, path-checks, property-checks, scalar-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
x <- as.Date("2000-01-01")
check_date(x)
check_date(1L) |> try()

class(x) <- c("my_date", class(x))
check_date(bare(x)) |> try()

x <- x + 1:5
check_date(x, n = 3) |> try()
check_date(x, n = at_least(10)) |> try()
check_date(x, n = at_most(3)) |> try()
check_date(x, n = in_range(6, 10)) |> try()

x <- data.frame(x = 1:3, y = 1:3)
check_tibble(x) |> try()

class(x) <- c("my_tbl", "tbl_df", class(x))
check_tibble(x)
check_tibble(bare(x)) |> try()

check_tibble(x, nrow = 2) |> try()
check_tibble(x, nrow = at_least(4)) |> try()
check_tibble(x, nrow = in_range(1, 2)) |> try()
check_tibble(x, ncol = 3) |> try()
check_tibble(x, ncol = at_most(1)) |> try()
check_tibble(x, ncol = in_range(3, 5)) |> try()
```

scalar-type-checks *Scalar type checks*

Description

Check if inputs are scalars of an expected type and throw an error if not.

Usage

```
check_scalar_list(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_atomic(
  x,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_vector(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_integer(
  x,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_integerish(
  x,
  ...,
  finite = FALSE,
```

```
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_double(
  x,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_complex(
  x,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_character(
  x,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_logical(
  x,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_raw(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

```
check_scalar_bytes(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_scalar_numeric(
  x,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
allow_na	Whether x is allowed to contain NA values.
finite	Whether x is required to contain only finite values (i.e. no NA, Inf, -Inf, or NaN).

Details

These functions can be used with the bare() modifier to check if an object is a bare R object (i.e. has no class attribute).

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

To handle empty strings ("") use check_string() instead of check_scalar_character().

These check functions are wrappers of their corresponding rlang functions. The exception is check_scalar_numeric(), which uses is.numeric().

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-value-checks, type-checks, walk-check

Examples

```
x <- 1L
check_scalar_integer(x)
check_scalar_double(x) |> try()

check_scalar_list(list(list()))
check_scalar_list(list(1, 2)) |> try()

check_scalar_character(NA_character_, allow_na = FALSE) |> try()
check_scalar_double(Inf, finite = TRUE) |> try()

check_scalar_logical(NULL, allow_null = TRUE)

x <- 1.0
check_scalar_integerish(x)
```

---

scalar-value-checks *Scalar value checks*

Description

Check if inputs are expected scalar values and throw an error if not.

Usage

```
check_true(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_false(
  x,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_bool(
  x,
```

```
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_string(
  x,
  ...,
  string = NULL,
  allow_empty = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
string	A character vector of allowed values for x. If NULL, the value is not checked. The check passes if x is **any** of the values in string.
allow_empty	Whether x is allowed to be an empty string (i.e. when FALSE, "" is not allowed).

Value

NULL invisibly if the check passes, otherwise an error is thrown.

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, type-checks, walk-check

Examples

```
x <- TRUE
check_true(x)
check_false(x) |> try()

check_bool(NA) |> try()
check_bool(NULL, allow_null = TRUE)
```

```
x <- "a"
check_string(x)
check_string(x, string = c("a", "b"))
check_string(x, string = c("b", "c")) |> try()
check_string("", allow_empty = FALSE) |> try()
```

---

type-checks *Type checks*

Description

Check if inputs are expected types and throw an error if not.

Usage

```
check_list(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_atomic(
  x,
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_vector(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_integer(
  x,
  n = NULL,
```

```
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_integerish(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_double(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_complex(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_character(
  x,
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_logical(
```

```
  x,
  n = NULL,
  ...,
  allow_na = TRUE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_raw(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_bytes(
  x,
  n = NULL,
  ...,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)

check_null(x, ..., arg = caller_arg(x), call = caller_env())

check_numeric(
  x,
  n = NULL,
  ...,
  finite = FALSE,
  allow_null = FALSE,
  arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	An object to check.
n	The expected length of x.
...	Additional arguments passed to cli_abort() which forwards unmatched arguments to abort().
allow_null	Whether x is allowed to be NULL.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.

call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
allow_na	Whether x is allowed to contain NA values.
finite	Whether x is required to contain only finite values (i.e. no NA, Inf, -Inf, or NaN).

Details

These functions can be used with the bare() modifier to check if an object is a bare R object (i.e. has no class attribute), and the length modifiers at_least(), at_most(), and in_range() to modify the behaviour of the length checking n argument.

Value

NULL invisibly if the check passes, otherwise an error is thrown.

Note

check_null() cannot use bare() since NULL cannot have a class attribute.

These check functions are wrappers of their corresponding rlang functions. The exception is check_numeric(), which uses is.numeric().

See Also

Other checks: array-type-checks, forbidden-value-checks, inheritance-checks, oop-checks, path-checks, property-checks, s3-type-checks, scalar-type-checks, scalar-value-checks, walk-check

Examples

```
x <- c(1, 2, 3)

check_integer(x) |> try()
check_integerish(x)
check_scalar_double(x) |> try()
check_double(x, n = 2) |> try()
check_double(x, n = at_least(4)) |> try()
check_double(x, n = at_most(2)) |> try()
check_double(x, n = in_range(1, 2)) |> try()

check_integer(bare(factor(1))) |> try()

check_double(c(1L, NA), allow_na = FALSE) |> try()
check_double(c(1.5, NA), finite = TRUE) |> try()

check_double(NULL, allow_null = TRUE)

# NULL list elements are not considered NULL
check_list(list(NULL), allow_null = TRUE) |> try()
```

walk-check *Apply a predicate check to each element of a vector*

Description

Apply a predicate check function to each element of a vector and throw an error if any element fails the check.

Usage

```
walk_check(.x, .f, ..., call = caller_env())
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function or formula to apply to each element of `.x` (passed to `as_function()`). Must return a logical vector of all `TRUE` for no error to occur. Non-`logical` and `NA` values will trigger an error.
`...`	Additional arguments passed to `cli_abort()` which forwards unmatched arguments to `abort()`.
`call`	The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the call argument of `abort()` for more information.

Details

`walk_check()` is designed to work with predicate functions, throwing an error indicating the element that fails the check. If you wish to use a function for `.f` that itself errors, pass contextual information to that function directly (e.g., using a shorthand anonymous function), as the error will be thrown from that function's context and won't have direct access to information from the caller such as `...` and `call`.

Value

`.x` invisibly if all checks pass, otherwise an error is thrown.

See Also

Other checks: `array-type-checks`, `forbidden-value-checks`, `inheritance-checks`, `oop-checks`, `path-checks`, `property-checks`, `s3-type-checks`, `scalar-type-checks`, `scalar-value-checks`, `type-checks`

Examples

```
x <- list(1, 2, "a")
walk_check(x, is.atomic)
walk_check(x, ~ length(.x) == 1L)
walk_check(x, is.numeric) |> try()
walk_check(x, \(el) nchar(el) == 1L)

# Named elements are shown in the error.
x <- list(a = 1, b = 2, c = "a")
walk_check(x, is.numeric) |> try()
```

Index