Package ‘mlr3misc’

January 11, 2022

Title Helper Functions for ‘mlr3’
Version 0.10.0
Description Frequently used helper functions and assertions used in 'mlr3' and its companion packages. Comes with helper functions for functional programming, for printing, to work with 'data.table', as well as some generally useful 'R6' classes. This package also supersedes the package 'BBmisc'.
License LGPL-3
BugReports https://github.com/mlr-org/mlr3misc/issues
Depends R (>= 3.1.0)
Imports backports (>= 0.1.5), checkmate, data.table, digest, R6
Suggests callr, evaluate, paradox, testthat (>= 3.0.0)
Encoding UTF-8
NeedsCompilation yes
Config/testthat/edition 3
Config/testthat/parallel true
RoxygenNote 7.1.2
Author Michel Lang [cre, aut] (<https://orcid.org/0000-0001-9754-0393>), Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>)
Maintainer Michel Lang <michellang@gmail.com>
Repository CRAN
Date/Publication 2022-01-11 14:52:41 UTC

R topics documented:

  mlr3misc-package ................................................. 3
  assert_ro_binding ................................................ 4
  as_factor ....................................................... 4
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_short_string</td>
<td>5</td>
</tr>
<tr>
<td>calculate_hash</td>
<td>6</td>
</tr>
<tr>
<td>capitalize</td>
<td>6</td>
</tr>
<tr>
<td>check_packages_installed</td>
<td>7</td>
</tr>
<tr>
<td>chunk_vector</td>
<td>8</td>
</tr>
<tr>
<td>compat-map</td>
<td>9</td>
</tr>
<tr>
<td>compose</td>
<td>11</td>
</tr>
<tr>
<td>compute_mode</td>
<td>12</td>
</tr>
<tr>
<td>count_missing</td>
<td>13</td>
</tr>
<tr>
<td>crate</td>
<td>13</td>
</tr>
<tr>
<td>cross_join</td>
<td>14</td>
</tr>
<tr>
<td>Dictionary</td>
<td>15</td>
</tr>
<tr>
<td>dictionary_sugar_get</td>
<td>18</td>
</tr>
<tr>
<td>did_you_mean</td>
<td>19</td>
</tr>
<tr>
<td>distinct_values</td>
<td>20</td>
</tr>
<tr>
<td>encapsulate</td>
<td>21</td>
</tr>
<tr>
<td>enframe</td>
<td>22</td>
</tr>
<tr>
<td>extract_vars</td>
<td>23</td>
</tr>
<tr>
<td>format_bib</td>
<td>24</td>
</tr>
<tr>
<td>formulate</td>
<td>25</td>
</tr>
<tr>
<td>get_private</td>
<td>25</td>
</tr>
<tr>
<td>get_seed</td>
<td>26</td>
</tr>
<tr>
<td>has_element</td>
<td>26</td>
</tr>
<tr>
<td>ids</td>
<td>27</td>
</tr>
<tr>
<td>insert_named</td>
<td>28</td>
</tr>
<tr>
<td>invoke</td>
<td>29</td>
</tr>
<tr>
<td>is_scalar_na</td>
<td>30</td>
</tr>
<tr>
<td>keep_in_bounds</td>
<td>30</td>
</tr>
<tr>
<td>leanify_r6</td>
<td>31</td>
</tr>
<tr>
<td>load_dataset</td>
<td>32</td>
</tr>
<tr>
<td>map_values</td>
<td>32</td>
</tr>
<tr>
<td>modify_if</td>
<td>33</td>
</tr>
<tr>
<td>named_list</td>
<td>34</td>
</tr>
<tr>
<td>named_vector</td>
<td>34</td>
</tr>
<tr>
<td>names2</td>
<td>35</td>
</tr>
<tr>
<td>open_help</td>
<td>36</td>
</tr>
<tr>
<td>printf</td>
<td>36</td>
</tr>
<tr>
<td>rcbind</td>
<td>37</td>
</tr>
<tr>
<td>rd_info</td>
<td>38</td>
</tr>
<tr>
<td>recycle_vectors</td>
<td>39</td>
</tr>
<tr>
<td>register_namespace_callback</td>
<td>39</td>
</tr>
<tr>
<td>reorder_vector</td>
<td>40</td>
</tr>
<tr>
<td>require_namespaces</td>
<td>41</td>
</tr>
<tr>
<td>rowwise_table</td>
<td>42</td>
</tr>
<tr>
<td>sequence_helpers</td>
<td>42</td>
</tr>
<tr>
<td>set_class</td>
<td>43</td>
</tr>
<tr>
<td>set_names</td>
<td>44</td>
</tr>
<tr>
<td>shuffle</td>
<td>45</td>
</tr>
</tbody>
</table>
mlr3misc-package

mlr3misc-package mlr3misc: Helper Functions for 'mlr3'

Description
Frequently used helper functions and assertions used in 'mlr3' and its companion packages. Comes with helper functions for functional programming, for printing, to work with 'data.table', as well as some generally useful 'R6' classes. This package also supersedes the package 'BBmisc'.

Author(s)
Maintainer: Michel Lang <michellang@gmail.com> (ORCID)
Authors:
• Patrick Schratz <patrick.schratz@gmail.com> (ORCID)

See Also
Useful links:
• https://mlr3misc.mlr-org.com
• https://github.com/mlr-org/mlr3misc
• Report bugs at https://github.com/mlr-org/mlr3misc/issues
assert_ro_binding  Assertion for Active Bindings in R6 Classes

Description
This assertion is intended to be called in active bindings of an R6::R6Class which does not allow assignment. If rhs is not missing, an exception is raised.

Usage
assert_ro_binding(rhs)

Arguments
rhs  (any)
If not missing, an exception is raised.

Value
Nothing.

as_factor  Convert to Factor

Description
Converts a vector to a factor() and ensures that levels are in the order of the provided levels.

Usage
as_factor(x, levels, ordered = is.ordered(x))

Arguments
x  (atomic vector())
Vector to convert to factor.
levels  (character())
Levels of the new factor.
ordered  (logical(1))
If TRUE, create an ordered factor.

Value
(factor()).
as_short_string

Examples

```r
x = factor(c("a", "b"))
y = factor(c("a", "b"), levels = c("b", "a"))

# x with the level order of y
as_factor(x, levels(y))

# y with the level order of x
as_factor(y, levels(x))
```

as_short_string

Convert R Object to a Descriptive String

Description

This function is intended to be convert any R object to a short descriptive string, e.g. in `base::print()` functions.

The following rules apply:

- if `x` is `atomic()` with length 0 or 1: printed as-is.
- if `x` is `atomic()` with length greater than 1, `x` is collapsed with ",", and the resulting string is truncated to `trunc_width` characters.
- if `x` is an expression: converted to character.
- Otherwise: the class is printed.

If `x` is a list, the above rules are applied (non-recursively) to its elements.

Usage

```r
as_short_string(x, width = 30L, num_format = "%4g")
```

Arguments

- `x` (any)
  Arbitrary object.
- `width` (integer(1))
  Truncate strings to width `width`.
- `num_format` (character(1))
  Used to format numerical scalars via `base::sprintf()`.

Value

(character(1)).

Examples

```r
as_short_string(list(a = 1, b = NULL, "foo", c = 1:10))
```
**calculate_hash**  
*Calculate a Hash for Multiple Objects*

**Description**

Calls `digest::digest()` to calculate the hash for all objects provided.

The following operations are performed to make hashing more robust:

- If an object is a `function()`, the formals and the body are hashed separately. This ensures that the bytecode or parent environment are not be included in the hash.
- If an object is a `data.table::data.table()`, the data.table is converted to a regular list. This ensures that keys and indices are not included in the hash.

Note that this only applies to top level objects, these transformations are not done recursively.

**Usage**

```
calculate_hash(...)  
```

**Arguments**

- `...` (any)  
  Objects to hash.

**Value**

(character(1)).

**Examples**

```
calculate_hash(iris, 1, "a")  
```

---

**capitalize**  
*Capitalize the First Letter of Strings*

**Description**

Takes a character vector and changes the first letter of each element to uppercase.

**Usage**

```
capitalize(str)  
```

**Arguments**

- `str` (character()).
check_packages_installed

Value

Character vector, same length as str.

Examples

capitalize("foo bar")

check_packages_installed

Check that packages are installed, without loading them

Description

Calls find.package() to check if the all packages are installed.

Usage

check_packages_installed(
  pkgs,
  warn = TRUE,
  msg = "The following packages are required but not installed: %s"
)

Arguments

pkgs (character())
Packages to check.

warn (logical(1))
If TRUE, signals a warning of class "packageNotFoundWarning" about the missing packages.

msg (character(1))
Format of the warning message. Use "%s" as placeholder for the list of packages.

Value

(logical()) named with package names. TRUE if the respective package is installed, FALSE otherwise.

Examples

check_packages_installed(c("mlr3misc", "foobar"), warn = FALSE)

# catch warning
tryCatch(check_packages_installed(c("mlr3misc", "foobaar")),
  packageNotFoundWarning = function(w) as.character(w))
chunk_vector

Description

Chunk atomic vectors into parts of roughly equal size. chunk() takes a vector length \( n \) and returns an integer with chunk numbers. chunk_vector() uses base::split() and chunk() to split an atomic vector into chunks.

Usage

chunk_vector(x, n_chunks = NULL, chunk_size = NULL, shuffle = TRUE)

chunk(n, n_chunks = NULL, chunk_size = NULL, shuffle = TRUE)

Arguments

- **x** (vector())
  Vector to split into chunks.
- **n_chunks** (integer(1))
  Requested number of chunks. Mutually exclusive with chunk_size and props.
- **chunk_size** (integer(1))
  Requested number of elements in each chunk. Mutually exclusive with n_chunks and props.
- **shuffle** (logical(1))
  If TRUE, permutes the order of x before chunking.
- **n** (integer(1))
  Length of vector to split.

Value

chunk() returns a integer() of chunk indices, chunk_vector() a list() of integer vectors.

Examples

```r
x = 1:11
ch = chunk(length(x), nChunks = 2)
table(ch)
split(x, ch)
chunk_vector(x, nChunks = 2)
chunk_vector(x, nChunks = 3, shuffle = TRUE)
```
compat-map

Description

Map-like functions, similar to the ones implemented in `purrr`:

- `map()` returns the results of `.f` applied to `.x` as list. If `.f` is not a function, `map` will call `[[` on all elements of `.x` using the value of `.f` as index.
- `imap()` applies `.f` to each value of `.x` (passed as first argument) and its name (passed as second argument). If `.x` does not have names, a sequence along `.x` is passed as second argument instead.
- `pmap()` expects `.x` to be a list of vectors of equal length, and then applies `.f` to the first element of each vector of `.x`, then the second element of `.x`, and so on.
- `map_if()` applies `.f` to each element of `.x` where the predicate `.p` evaluates to `TRUE`.
- `map_at()` applies `.f` to each element of `.x` referenced by `.at`. All other elements remain unchanged.
- `keep()` keeps those elements of `.x` where predicate `.p` evaluates to `TRUE`.
- `discard()` discards those elements of `.x` where predicate `.p` evaluates to `TRUE`.
- `every()` is `TRUE` if predicate `.p` evaluates to `TRUE` for each `.x`.
- `some()` is `TRUE` if predicate `.p` evaluates to `TRUE` for at least one `.x`.
- `detect()` returns the first element where predicate `.p` evaluates to `TRUE`.
- `walk()`, `iwalk()` and `pwalk()` are the counterparts to `map()`, `imap()` and `pmap()`, but just visit (or change by reference) the elements of `.x`. They return input `.x` invisibly.

Additionally, the functions `map()`, `imap()` and `pmap()` have type-safe variants with the following suffixes:

- `*_lgl()` returns a `logical(length(.x))`.
- `*_int()` returns a `integer(length(.x))`.
- `*_dbl()` returns a `double(length(.x))`.
- `*_chr()` returns a `character(length(.x))`.
- `*_br()` returns an object where the results of `.f` are put together with `base::rbind()`.
- `*_bc()` returns an object where the results of `.f` are put together with `base::cbind()`.
- `*_dtr()` returns a `data.table::data.table()` where the results of `.f` are put together in a `base::rbind()` fashion.
- `*_dtc()` returns a `data.table::data.table()` where the results of `.f` are put together in an `base::cbind()` fashion.
Usage

map(.x, .f, ...)
map_lgl(.x, .f, ...)
map_int(.x, .f, ...)
map_dbl(.x, .f, ...)
map_chr(.x, .f, ...)
map_br(.x, .f, ...)
map_bc(.x, .f, ...)
map_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
map_dtc(.x, .f, ...)
pmap(.x, .f, ...)
pmap_lgl(.x, .f, ...)
pmap_int(.x, .f, ...)
pmap_dbl(.x, .f, ...)
pmap_chr(.x, .f, ...)
pmap_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
pmap_dtc(.x, .f, ...)
imap(.x, .f, ...)
imap_lgl(.x, .f, ...)
imap_int(.x, .f, ...)
imap_dbl(.x, .f, ...)
imap_chr(.x, .f, ...)
imap_dtr(.x, .f, ..., .fill = FALSE, .idcol = NULL)
imap_dtc(.x, .f, ...)
keep(.x, .f, ...)
compose

## Description

Composes two or more functions into a single function. The returned function calls all provided functions in reverse order: The return value of the last function servers as input for the next to last function, and so on.

### Arguments

- `.x` (list() | atomic vector()).
- `.f` (function() | character() | integer())
  
  Function to apply, or element to extract by name (if `.f` is character()) or position (if `.f` is integer()).

- `...` (any)
  
  Additional arguments passed down to `.f` or `.p`.

- `.fill` (logical(1))
  
  Passed down to `data.table::rbindlist()`.

- `.idcol` (logical(1))
  
  Passed down to `data.table::rbindlist()`.

- `.p` (function() | logical())
  
  Predicate function.

- `.at` (character() | integer() | logical())
  
  Index vector.
Usage

compose(...)

Arguments

... (functions)
Functions to compose.

Value

(function()) which calls the functions provided via ... in reverse order.

Examples

f = compose(function(x) x + 1, function(x) x / 2)
f(10)

---

**compute_mode**

*Compute The Mode*

**Description**

Computes the mode (most frequent value) of an atomic vector.

**Usage**

compute_mode(x, ties_method = "random", na_rm = TRUE)

**Arguments**

x (vector()).

<table>
<thead>
<tr>
<th>ties_method</th>
</tr>
</thead>
<tbody>
<tr>
<td>character(1)</td>
</tr>
<tr>
<td>Handling of ties. One of &quot;first&quot;, &quot;last&quot; or &quot;random&quot; to return the first tied value, the last tied value, or a randomly selected tied value, respectively.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>na_rm</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical(1)</td>
</tr>
<tr>
<td>If TRUE, remove missing values prior to computing the mode.</td>
</tr>
</tbody>
</table>

**Value**

(vector(1)): mode value.

**Examples**

compute_mode(c(1, 1, 1, 2, 2, 3))
compute_mode(c(1, 1, 1, 2, 2, 3), ties_method = "last")
compute_mode(c(1, 1, 1, 2, 2, 3), ties_method = "random")
count_missing

Count Missing Values in a Vector Same as `sum(is.na(x))`, but without the allocation.

Description

Count Missing Values in a Vector
Same as `sum(is.na(x))`, but without the allocation.

Usage

count_missing(x)

Arguments

x vector()
Supported are logical, integer, double, complex and string vectors.

Value

(integer(1)) number of missing values.

Examples

count_missing(c(1, 2, NA, 4, NA))

crate

Isolate a Function from its Environment

Description

Put a function in a "lean" environment that does not carry unnecessary baggage with it (e.g. references to datasets).

Usage

crate(.fn, ..., .parent = .GlobalEnv)

Arguments

.fn (function())
function to crate

...(any)
The objects, which should be visible inside .fn.

.parent (environment)
Parent environment to look up names. Default so the global environment.
Examples

```r
meta_f = function(z) {
  x = 1
  y = 2
  crate(function() {
    c(x, y, z)
  }, x)
  x = 100
  y = 200
  z = 300
  f = meta_f(1)
  f()
}
```

cross_join

Cross-Join for data.table

Description

A safe version of `data.table::CJ()` in case a column is called `sorted` or `unique`.

Usage

cross_join(dots, sorted = TRUE, unique = FALSE)

Arguments

dots (named list())
  Vectors to cross-join.

sorted (logical(1))
  See `data.table::CJ()`.

unique (logical(1))
  See `data.table::CJ()`.

Value

data.table().

Examples

cross_join(dots = list(sorted = 1:3, b = letters[1:2]))
Description

A key-value store for R6::R6 objects. On retrieval of an object, the following applies:

- If the object is a R6ClassGenerator, it is initialized with new().
- If the object is a function, it is called and must return an instance of a R6::R6 object.
- If the object is an instance of a R6 class, it is returned as-is.

Default argument required for construction can be stored alongside their constructors by passing them to $add().

S3 methods

- as.data.table(d)
  Dictionary -> data.table::data.table()
  Converts the dictionary to a data.table::data.table().

Public fields

items (environment())
  Stores the items of the dictionary

Methods

Public methods:

- Dictionary$new()
- Dictionary$format()
- Dictionary$print()
- Dictionary$keys()
- Dictionary$has()
- Dictionary$get()
- Dictionary$mget()
- Dictionary$add()
- Dictionary$remove()
- Dictionary$required_args()
- Dictionary$clone()

Method new(): Construct a new Dictionary.

Usage:
Dictionary$new()

Method format(): Format object as simple string.
**Usage:**
Dictionary$format()

**Method** print(): Print object.

**Usage:**
Dictionary$print()

**Method** keys(): Returns all keys which comply to the regular expression pattern. If pattern is NULL (default), all keys are returned.

**Usage:**
Dictionary$keys(pattern = NULL)

**Arguments:**
pattern (character(1)).

**Returns:** character() of keys.

**Method** has(): Returns a logical vector with TRUE at its i-th position if the i-th key exists.

**Usage:**
Dictionary$has(keys)

**Arguments:**
keys (character()).

**Returns:** logical().

**Method** get(): Retrieves object with key key from the dictionary. Additional arguments must be named and are passed to the constructor of the stored object.

**Usage:**
Dictionary$get(key, ...)

**Arguments:**
key (character(1)).
... (any)
   Passed down to constructor.

**Returns:** Object with corresponding key.

**Method** mget(): Returns objects with keys keys in a list named with keys. Additional arguments must be named and are passed to the constructors of the stored objects.

**Usage:**
Dictionary$mget(keys, ...)

**Arguments:**
keys (character()).
... (any)
   Passed down to constructor.

**Returns:** Named list() of objects with corresponding keys.
**Method add():** Adds object value to the dictionary with key key, potentially overwriting a previously stored item. Additional arguments in ... must be named and are passed as default arguments to value during construction. The names of all additional arguments which are mandatory for construction and missing in ... should be listed in required_args.

*Usage:*
Dictionary$add(key, value, ..., required_args = character())

*Arguments:*
key (character(1)).
value (any).
... (any)
   Passed down to constructor.
required_args (character()).

*Returns:* Dictionary.

**Method remove():** Removes objects with from the dictionary.

*Usage:*
Dictionary$remove(keys)

*Arguments:*
keys (character())
   Keys of objects to remove.

*Returns:* Dictionary.

**Method required_args():** Returns the names of arguments required to construct the object.

*Usage:*
Dictionary$required_args(key)

*Arguments:*
key (character(1))
   Key of object to query for required arguments.

*Returns:* character() of names of required arguments.

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*
Dictionary$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

**Examples**

```r
library(R6)
item1 = R6Class("Item", public = list(x = 1))
item2 = R6Class("Item", public = list(x = 2))
d = Dictionary$new()
d$add("a", item1)
d$add("b", item2)
```
A Quick Way to Initialize Objects from Dictionaries

Description

Given a Dictionary, retrieve objects with provided keys.

- `dictionary_sugar_get()` to retrieve a single object with key `.key`.
- `dictionary_sugar_mget()` to retrieve a list of objects with keys `.keys`.
- `dictionary_sugar()` is deprecated in favor of `dictionary_sugar_get()`.
- If `.key` or `.keys` is missing, the dictionary itself is returned.

Arguments in `...` must be named and are consumed in the following order:

1. All arguments whose names match the name of an argument of the constructor are passed to the $get() method of the Dictionary for construction.
2. All arguments whose names match the name of a parameter of the paradox::ParamSet of the constructed object are set as parameters. If there is no paradox::ParamSet in obj$param_set, this step is skipped.
3. All remaining arguments are assumed to be regular fields of the constructed R6 instance, and are assigned via `<-`.

Usage

```r
dictionary_sugar_get(dict, .key, ...)
dictionary_sugar(dict, .key, ...)
dictionary_sugar_mget(dict, .keys, ...)
```

Arguments

- **dict** (Dictionary).
- **.key** (character(1))
  Key of the object to construct.
- **...** (any)
  See description.
- **.keys** (character())
  Keys of the objects to construct.
Value

R6::R6Class()

Examples

library(R6)
item = R6Class("Item", public = list(x = 0))
d = Dictionary$new()
d$add("key", item)
dictionary_sugar_get(d, "key", x = 2)

Description

Helps to suggest alternatives from a list of strings, based on the string similarity in utils::adist().

Usage

did_you_mean(str, candidates)

Arguments

str (character(1)) String.
candidates (character()) Candidate strings.

Value

(character(1)). Either a phrase suggesting one or more candidates from candidates, or an empty string if no close match is found.

Examples

did_you_mean("yep", c("yes", "no"))
## distinct_values

### Get Distinct Values

**Description**

Extracts the distinct values of an atomic vector, with the possibility to drop levels and remove missing values.

**Usage**

```r
distinct_values(x, drop = TRUE, na_rm = TRUE)
```

**Arguments**

- **x** (atomic vector()).
- **drop** :: logical(1)
  
  If TRUE, only returns values which are present in `x`. If FALSE, returns all levels for `factor()` and `ordered()`, as well as TRUE and FALSE for `logical()`s.
- **na_rm** :: logical(1)
  
  If TRUE, missing values are removed from the vector of distinct values.

**Value**

(atomic vector()) with distinct values in no particular order.

**Examples**

```r
# for factors:
x = factor(c(letters[1:2], NA), levels = letters[1:3])
distinct_values(x)
distinct_values(x, na_rm = FALSE)
distinct_values(x, drop = FALSE)
distinct_values(x, drop = FALSE, na_rm = FALSE)

# for logsicals:
distinct_values(TRUE, drop = FALSE)

# for numerics:
distinct_values(sample(1:3, 10, replace = TRUE))
```
Encapsulate Function Calls for Logging

Description

Evaluates a function while both recording an output log and measuring the elapsed time. There are currently three different modes implemented to encapsulate a function call:

- "none": Just runs the call in the current session and measures the elapsed time. Does not keep a log, output is printed directly to the console. Works well together with `traceback()`.
- "evaluate": Uses the package `evaluate` to call the function, measure time and do the logging.
- "callr": Uses the package `callr` to call the function, measure time and do the logging. This encapsulation spawns a separate R session in which the function is called. While this comes with a considerable overhead, it also guards your session from being teared down by segfaults.

Usage

```r
encapsulate(
  method,
  .f,
  .args = list(),
  .opts = list(),
  .pkgs = character(),
  .seed = NA_integer_,
  .timeout = Inf
)
```

Arguments

- `method` (character(1))
  One of "none", "evaluate" or "callr".
- `f` (function())
  Function to call.
- `args` (list())
  Arguments passed to `.f`.
- `opts` (named list())
  Options for the function call. Options get reset on exit.
- `pkgs` (character())
  Packages to load (not attach).
- `seed` (integer(1))
  Random seed to set before invoking the function call. Gets reset to the previous seed on exit.
- `timeout` (numeric(1))
  Timeout in seconds. Uses `setTimeLimit()` for "none" and "evaluate" encapsulation. For "callr" encapsulation, the timeout is passed to `callr::r()`.
Value

(named list()) with three fields:

- "result": the return value of .f
- "elapsed": elapsed time in seconds. Measured as proc.time() difference before/after the function call.
- "log": data.table() with columns "class" (ordered factor with levels "output", "warning" and "error") and "message" (character()).

Examples

```r
f = function(n) {
  message("hi from f")
  if (n > 5) {
    stop("n must be <= 5")
  }
  runif(n)
}
encapsulate("none", f, list(n = 1), .seed = 1)

if (requireNamespace("evaluate", quietly = TRUE)) {
  encapsulate("evaluate", f, list(n = 1), .seed = 1)
}
if (requireNamespace("callr", quietly = TRUE)) {
  encapsulate("callr", f, list(n = 1), .seed = 1)
}
```

---

enframe

*Convert a Named Vector into a data.table and Vice Versa*

**Description**

enframe() returns a data.table::data.table() with two columns: The names of x (or seq_along(x) if unnamed) and the values of x.

deframe() converts a two-column data.frame to a named vector. If the data.frame only has a single column, an unnamed vector is returned.

**Usage**

```r
enframe(x, name = "name", value = "value")
deframe(x)
```
extract_vars

Arguments

\[ x \]
- (vector() (enframe()) or data.frame() (deframe()))
  Vector to convert to a data.table::data.table().

\[ \text{name} \]
- (character(1))
  Name for the first column with names.

\[ \text{value} \]
- (character(1))
  Name for the second column with values.

Value

data.table::data.table() or named vector.

Examples

\[ x = 1:3 \]
\[ \text{enframe}(x) \]

\[ x = \text{set_names}(1:3, \text{letters}[1:3]) \]
\[ \text{enframe}(x, \text{value} = "x\_values") \]

extract_vars  Extract Variables from a Formula

Description

Given a formula() \( f \), returns all variables used on the left-hand side and right-hand side of the formula.

Usage

extract_vars(f)

Arguments

\[ f \]  (formula()).

Value

(list()) with elements "lhs" and "rhs", both character().

Examples

extract_vars(Species \sim Sepal.Width + Sepal.Length)
extract_vars(Species \sim .)
Format Bibentries in Roxygen

Description

Operates on a named list of `bibentry()` entries and formats them nicely for documentation with `roxygen2`.

- `format_bib()` is intended to be called in the @references section and prints the complete entry using `toRd()`.
- `cite_bib()` returns the family name of the first author (if available, falling back to the complete author name if not applicable) and the year in format “[LastName] (YYYY)”.

Usage

```r
format_bib(..., bibentries = NULL, envir = parent.frame())
cite_bib(..., bibentries = NULL, envir = parent.frame())
```

Arguments

- `...` (character())
  One or more names of bibentries.
- `bibentries` (named list())
  Named list of bibentries.
- `envir` (environment)
  Environment to lookup bibentries if not provided.

Value

(character(1)).

Examples

```r
bibentries = list(checkmate = citation("checkmate"), R = citation())
format_bib("checkmate")
format_bib("R")
cite_bib("checkmate")
cite_bib("checkmate", "R")
```
formulate

Create Formulas

Description

Given the left-hand side and right-hand side as character vectors, generates a new `stats::formula()`.

Usage

`formulate(lhs = character(), rhs = character(), env = NULL, quote = "right")`

Arguments

- `lhs` (character())
  Left-hand side of formula. Multiple elements will be collapsed with " + ".
- `rhs` (character())
  Right-hand side of formula. Multiple elements will be collapsed with " + ".
- `env` (environment())
  Environment for the new formula. Defaults to NULL.
- `quote` (character(1))
  Which side of the formula to quote? Subset of ("left", "right"), defaulting to "right".

Value

`stats::formula()`

Examples

```r
formulate("Species", c("Sepal.Length", "Sepal.Width"))
formulate(rhs = c("Sepal.Length", "Sepal.Width"))
```

get_private

Extract Private Fields of R6 Objects

Description

Provides access to the private members of `R6::R6Class` objects.

Usage

`get_private(x)`
Arguments
x (any)
Object to extract the private members from.

Value
environment() of private members, or NULL if x is not an R6 object.

Examples
library(R6)
item = R6Class("Item", private = list(x = 1))$new()
get_private(item)$x

get_seed Get the Random Seed

Description
Retrieves the current random seed (.Random.seed in the global environment), and initializes the RNG first, if necessary.

Usage
get_seed()

Value
integer(). Depends on the base::RNGkind().

Examples
str(get_seed())

has_element Check if an Object is Element of a List

Description
Simply checks if a list contains a given object.

- NB1: Objects are compared with identity.
- NB2: Only use this on lists with complex objects, for simpler structures there are faster operations.
- NB3: Clones of R6 objects are not detected.
**ids**

Usage

\[
\text{has\_element}(.x, .y)
\]

Arguments

\[
.x \quad \text{(list() | atomic vector()).}
\]

\[
.y \quad \text{any}
\]

Object to test for.

Examples

\[
\text{has\_element(list(1, 2, 3), 1)}
\]

**ids**

*Extract ids from a List of Objects*

Description

None.

Usage

\[
\text{ids}(\text{xs})
\]

Arguments

\[
\text{xs} \quad \text{(list())}
\]

Every element must have a slot ‘id’.

Value

\[
\text{(character())}
\]

Examples

\[
\text{xs = list(a = list(id = “foo”, a = 1), bar = list(id = “bar”, a = 2))}
\]

\[
\text{ids}(\text{xs})
\]
Insert or Remove Named Elements

Insert elements from \( y \) into \( x \) by name, or remove elements from \( x \) by name. Works for vectors, lists, environments and data frames and data tables. Objects with reference semantic (\texttt{environment()}) and \texttt{data.table::data.table()} might be modified in-place.

### Usage

\begin{verbatim}
insert_named(x, y)
## Default S3 method:
insert_named(x, y)

## S3 method for class 'environment'
insert_named(x, y)

## S3 method for class 'data.frame'
insert_named(x, y)

## S3 method for class 'data.table'
insert_named(x, y)

remove_named(x, nn)

## S3 method for class 'environment'
remove_named(x, nn)

## S3 method for class 'data.frame'
remove_named(x, nn)

## S3 method for class 'data.table'
remove_named(x, nn)
\end{verbatim}

### Arguments

- **\( x \)**: \( \text{vector()} \mid \text{list()} \mid \text{environment()} \mid \text{data.table::data.table()} \)
  
  Object to insert elements into, or remove elements from. Changes are by-reference for environments and data tables.

- **\( y \)**: \( \text{list()} \)
  
  List of elements to insert into \( x \).

- **\( nn \)**: \( \text{character()} \)
  
  Character vector of elements to remove.
Value

Modified object.

Examples

```r
x = list(a = 1, b = 2)
insert_named(x, list(b = 3, c = 4))
remove_named(x, "b")
```

**Description**

An alternative interface for `do.call()`, similar to the deprecated function in `purrr`. This function tries hard to not evaluate the passed arguments too eagerly which is important when working with large R objects.

It is recommended to pass all arguments named in order to not rely on positional argument matching.

**Usage**

```r
invoke(
  .f,
  ..., 
  .args = list(),
  .opts = list(),
  .seed = NA_integer_,
  .timeout = Inf
)
```

**Arguments**

- `.f` (function())
  Function to call.
- `...` (any)
  Additional function arguments passed to `.f`.
- `.args` (list())
  Additional function arguments passed to `.f`, as (named) `list()`. These arguments will be concatenated to the arguments provided via `...`.
- `.opts` (named `list()`)  
  List of options which are set before the `.f` is called. Options are reset to their previous state afterwards.
- `.seed` (integer())
  Random seed to set before invoking the function call. Gets reset to the previous seed on exit.
- `.timeout` (numeric(1))
  Timeout in seconds. Uses `setTimeLimit()`. Note that timeouts are only triggered on a user interrupt, not in compiled code.
Examples

invoke(mean, .args = list(x = 1:10))
invoke(mean, na.rm = TRUE, .args = list(1:10))

is_scalar_na  
Check for a Single Scalar Value

Description
Check for a Single Scalar Value

Usage

is_scalar_na(x)

Arguments

x (any)
Argument to check.

Value
(logical(1)).

keep_in_bounds  
Remove All Elements Out Of Bounds

Description
Filters vector x to only keep elements which are in bounds [lower, upper]. This is equivalent to the following, but tries to avoid unnecessary allocations:

x[!is.na(x) & x >= lower & x <= upper]

Currently only works for integer x.

Usage

keep_in_bounds(x, lower, upper)

Arguments

x (integer())
Vector to filter.
lower (integer(1))
Lower bound.
upper (integer(1))
Upper bound.
**leanify_r6**

**Value**

(integer()) with only values in [lower, upper].

**Examples**

```r
keep_in_bounds(sample(20), 5, 10)
```

---

**Description**

`leanify_r6` moves the content of an `R6::R6Class`'s functions to an environment, usually the package's namespace, to save space during serialization of R6 objects. `leanify_package` move all methods of all R6 Classes to an environment.

The function in the class (i.e. the object generator) is replaced by a stump function that does nothing except calling the original function that now resides somewhere else.

It is possible to call this function after the definition of an `R6::R6` class inside a package, but it is preferred to use `leanify_package()` to just leanify all `R6::R6` classes inside a package.

**Usage**

```r
leanify_r6(cls, env = cls$parent_env)

leanify_package(pkg_env = parent.frame(), skip_if = function(x) FALSE)
```

**Arguments**

- `cls`: (R6::R6Class) Class generator to modify.
- `env`: (environment) The target environment where the function should be stored. This should be either `cls$parent_env` (default) or one of its parent environments, otherwise the stump function will not find the moved (original code) function.
- `pkg_env`: :: environment The namespace from which to leanify all R6 classes. Does not have to be a package namespace, but this is the intended usecase.
- `skip_if`: :: function Function with one argument: Is called for each individual `R6::R6Class`. If it returns TRUE, the class is skipped. Default function evaluating to FALSE always (i.e. skipping no classes).

**Value**

NULL.
load_dataset  

*Retrieve a Single Data Set*

**Description**

Loads a data set with name `id` from package `package` and returns it. If the package is not installed, an error with condition "packageNotFoundError" is raised. The name of the missing packages is stored in the condition as `packages`.

**Usage**

```r
load_dataset(id, package, keep_rownames = FALSE)
```

**Arguments**

- `id` (character(1))
  - Name of the data set.
- `package` (character(1))
  - Package to load the data set from.
- `keep_rownames` (logical(1))
  - Keep possible row names (default: FALSE).

**Examples**

```r
head(load_dataset("iris", "datasets"))
```

---

map_values  

*Replace Elements of Vectors with New Values*

**Description**

Replaces all values in `x` which match `old` with values in `new`. Values are matched with `base::match()`.

**Usage**

```r
map_values(x, old, new)
```

**Arguments**

- `x` (vector()).
- `old` (vector())
  - Vector with values to replace.
- `new` (vector())
  - Values to replace with. Will be forced to the same length as `old` with `base::rep_len()`.
modify_if

Value

(vector()) of the same length as x.

Examples

x = letters[1:5]

# replace all "b" with "_b_", and all "c" with "_c_"
old = c("b", "c")
new = c("_b_", "_c_")
map_values(x, old, new)

modify_if

Selectively Modify Elements of a Vector

Description

Modifies elements of a vector selectively, similar to the functions in purrr.
modify_if() applies a predicate function .p to all elements of .x and applies .f to those elements of .x where .p evaluates to TRUE.
modify_at() applies .f to those elements of .x selected via .at.

Usage

modify_if(.x, .p, .f, ...)
modify_at(.x, .at, .f, ...)

Arguments

.x (vector()).
.p (function())
    Predicate function.
.f (function())
    Function to apply on .x.
... (any)
    Additional arguments passed to .f.
.at ((integer() | character()))
    Index vector to select elements from .x.

Examples

x = modify_if(iris, is.factor, as.character)
str(x)

x = modify_at(iris, 5, as.character)
x = modify_at(iris, "Sepal.Length", sqrt)
str(x)
named_list  

Create a Named List

Description
Create a Named List

Usage
named_list(nn = character(0L), init = NULL)

Arguments

<table>
<thead>
<tr>
<th>nn</th>
<th>(character())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names of new list.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init</th>
<th>(any)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All list elements are initialized to this value.</td>
<td></td>
</tr>
</tbody>
</table>

Value
(named list()).

Examples

named_list(c("a", "b"))
named_list(c("a", "b"), init = 1)

named_vector  

Create a Named Vector

Description

Creates a simple atomic vector with init as values.

Usage
named_vector(nn = character(0L), init = NA)

Arguments

<table>
<thead>
<tr>
<th>nn</th>
<th>(character())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names of new vector</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init</th>
<th>(atomic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All vector elements are initialized to this value.</td>
<td></td>
</tr>
</tbody>
</table>
names2

Value
(named vector()).

Examples
```
named_vector(c("a", "b"), NA)
named_vector(character())
```

Description
A simple wrapper around `base::names()`. Returns a character vector even if no names attribute is set. Values NA and "" are treated as missing and replaced with the value provided in `missing_val`.

Usage
```
names2(x, missing_val = NA_character_)
```

Arguments
```
x (any)
   Object.

missing_val (atomic(1))
   Value to set for missing names. Default is `NA_character_`.
```

Value
(character(length(x))).

Examples
```
x = 1:3
names(x)
names2(x)

names(x)[1:2] = letters[1:2]
names(x)
names2(x, missing_val = "")
```
open_help  

*Opens a Manual Page*

---

**Description**

Simply opens a manual page specified in "package::topic" syntax.

**Usage**

```r
open_help(man)
```

**Arguments**

- `man` *(character(1))*

  Manual page to open in "package::topic" syntax.

**Value**

Nothing.

---

**printf**  

*Functions for Formatted Output and Conditions*

---

**Description**

`catf()`, `messagef()`, `warningf()` and `stopf()` are wrappers around `base::cat()`, `base::message()`, `base::warning()` and `base::stop()`, respectively. The call is not included for warnings and errors.

**Usage**

```r
catf(msg, ..., file = "", wrap = FALSE)
messagef(msg, ..., wrap = FALSE)
warningf(msg, ..., wrap = FALSE)
stopf(msg, ..., wrap = FALSE)
```
**rcbind**

**Bind Columns by Reference**

**Description**

Performs `base::cbind()` on data.tables, possibly by reference.

**Usage**

`rcbind(x, y)`

**Arguments**

- **x** 
  
  (data.table::data.table())
  
  data.table::data.table() to add columns to.

- **y** 
  
  (data.table::data.table())
  
  data.table::data.table() to take columns from.

**Value**

(data.table::data.table()): Updated x.
rd_info

## Examples

```r
x = data.table::data.table(a = 1:3, b = 3:1)
y = data.table::data.table(c = runif(3))
rcbind(x, y)
```

## Description

`rd_info()` is an internal generic to generate Rd or markdown code to be used in manual pages. `rd_format_string()` and `rd_format_range()` are string functions to assist generating proper Rd code.

## Usage

```r
rd_info(obj, ...) 
rd_format_range(lower, upper) 
rd_format_string(str, quote = c("\dQuote{", "}")) 
rd_format_packages(packages)
```

## Arguments

- **obj** *(any)*
  - Object of the respective class.

- **...** *(any)*
  - Additional arguments.

- **lower** *(numeric(1))*
  - Lower bound.

- **upper** *(numeric(1))*
  - Upper bound.

- **str** *(character())*
  - Vector of strings.

- **quote** *(character())*
  - Quotes to use around each element of `x`. Will be replicated to length 2.

- **packages** *(character())*
  - Vector of package names.

## Value

`character()`, possibly with markdown code.
**recycle_vectors**

*Recycle List of Vectors to Common Length*

**Description**

Repeats all vectors of a list .x to the length of the longest vector using `rep()` with argument `length.out`. This operation will only work if the length of the longest vectors is an integer multiple of all shorter vectors, and will throw an exception otherwise.

**Usage**

```r
recycle_vectors(.x)
```

**Arguments**

- `.x` (list).

**Value**

(list()) with vectors of same size.

**Examples**

```r
recycle_vectors(list(a = 1:3, b = 2))
```

---

**register_namespace_callback**

*Registers a Callback on Namespace load/unLoad Events*

**Description**

Register a function callback to be called after a namespace is loaded. Calls callback once if the namespace has already been loaded before and also adds an unload-hook that removes the load hook.

**Usage**

```r
register_namespace_callback(pkgname, namespace, callback)
```

**Arguments**

- `.pkgname` (character(1))
  
  Name of the package which registers the callback.

- `.namespace` (character(1))
  
  Namespace to react on.

- `.callback` (function())
  
  Function to call on namespace load.
Value

NULL.

---

reorder_vector  Reorder Vector According to Second Vector

Description

Returns an integer vector to order vector x according to vector y.

Usage

reorder_vector(x, y, na_last = NA)

Arguments

x  (vector()).
y  (vector()).
na_last  (logical(1))
What to do with values in x which are not in y?
• NA: Extra values are removed.
• FALSE: Extra values are moved to the beginning of the new vector.
• TRUE: Extra values are moved to the end of the new vector.

Value

(integer()).

Examples

# x subset of y
x = c("b", "a", "c", "d")
y = letters
x[reorder_vector(x, y)]

# y subset of x
y = letters[1:3]
x[reorder_vector(x, y)]
x[reorder_vector(x, y, na_last = TRUE)]
x[reorder_vector(x, y, na_last = FALSE)]
require_namespaces  

Require Multiple Namespaces

Description

Packages are loaded (not attached) via `base::requireNamespace()`. If at least one package cannot be loaded, an exception of class "packageNotFoundError" is raised. The character vector of missing packages is stored in the condition as `packages`.

Usage

```r
require_namespaces(
  pkgs,
  msg = "The following packages could not be loaded: %s",
  quietly = FALSE
)
```

Arguments

- **pkgs** (character())
  Packages to load.
- **msg** (character(1))
  Message to print on error. Use "%s" as placeholder for the list of packages.
- **quietly** (logical(1))
  If TRUE then returns TRUE if all packages available, otherwise FALSE.

Value

(named character()) of loaded packages (invisibly).

Examples

```r
require_namespaces("mlr3misc")

# catch condition, return missing packages
tryCatch(require_namespaces(c("mlr3misc", "foobaaar")),
  packageNotFoundError = function(e) e$packages)
```
**rowwise_table**  
*Row-Wise Constructor for 'data.table'*

**Description**

Similar to the *tibble* function *tribble()*, this function allows to construct tabular data in a row-wise fashion.

The first arguments passed as formula will be interpreted as column names. The remaining arguments will be put into the resulting table.

**Usage**

```r
rowwise_table(..., .key = NULL)
```

**Arguments**

- `...` (any)  
  Arguments: Column names in first rows as formulas (with empty left hand side), then the tabular data in the following rows.

- `.key` (character(1))  
  If not NULL, set the key via `data.table::setkeyv()` after constructing the table.

**Value**

`data.table::data.table()`.

**Examples**

```r
rowwise_table(
  ~a, ~b,
  1, "a",
  2, "b"
)
```

---

**sequence_helpers**  
*Sequence Construction Helpers*

**Description**

`seq_row()` creates a sequence along the number of rows of `x`, `seq_col()` a sequence along the number of columns of `x`. `seq_len()` and `seq_along()` are the 0-based counterparts to `base::seq_len()` and `base::seq_along()`.
**set_class**

**Usage**

```
seq_row(x)
seq_col(x)
seq_len0(n)
seq_along0(x)
```

**Arguments**

- `x` *(any)*
  - Arbitrary object. Used to query its rows, cols or length.
- `n` *(integer(1))*
  - Length of the sequence.

**Examples**

```
seq_len0(3)
```

---

**set_class**

**Set the Class**

**Description**

Simple wrapper for `class(x) = classes`.

**Usage**

```
set_class(x, classes)
```

**Arguments**

- `x` *(any)*
- `classes` *(character(1))*
  - Vector of new class names.

**Value**

Object `x`, with updated class attribute.

**Examples**

```
set_class(list(), c("foo1", "foo2"))
```
**set_names**

*Set Names*

**Description**

Sets the names (or colnames) of `x` to `nm`. If `nm` is a function, it is used to transform the already existing names of `x`.

**Usage**

```r
set_names(x, nm = x, ...)
set_col_names(x, nm, ...)
```

**Arguments**

- `x` *(any.)*
  Object to set names for.
- `nm` *(character() | function())*
  New names, or a function which transforms already existing names.
- `...` *(any)*
  Passed down to `nm` if `nm` is a function.

**Value**

`x` with updated names.

**Examples**

```r
x = letters[1:3]

# name x with itself:
x = set_names(x)
print(x)

# convert names to uppercase
x = set_names(x, toupper)
print(x)
```
shuffle  

Safe Version of Sample

Description
A version of sample() which does not treat positive scalar integer x differently. See example.

Usage
shuffle(x, n = length(x), ...)

Arguments
- x (vector()): Vector to sample elements from.
- n (integer()): Number of elements to sample.
- ... (any): Arguments passed down to base::sample.int().

Examples
x = 2:3
sample(x)
shuffle(x)

x = 3
sample(x)
shuffle(x)

str Collapse  

Collapse Strings

Description
Collapse multiple strings into a single string.

Usage
str Collapse(str, sep = "", quote = character(), n = Inf, ellipsis = "[...]")
Arguments

str (character())
Vector of strings.

sep (character(1))
String used to collapse the elements of x.

quote (character())
Quotes to use around each element of x. Will be replicated to length 2.

n (integer(1))
Number of elements to keep from x. See utils::head().

ellipsis (character(1))
If the string has to be shortened, this is signaled by appending ellipsis to str. Default is "[...]".

Value

(character(1)).

Examples

strCollapse(letters, quote = '/', n = 5)

---

str_indent  Indent Strings

Description

Formats a text block for printing.

Usage

str_indent(initial, str, width = 0.9 * getOption("width"), exdent = 2L, ...)

Arguments

initial (character(1))
Initial string, passed to strwrap().

str (character())
Vector of strings.

width (integer(1))
Width of the output.

exdent (integer(1))
Indentation of subsequent lines in paragraph.

... (any)
Additional parameters passed to strCollapse().
**Description**

`str_trunc()` truncates a string to a given width.

**Usage**

```r
str_trunc(str, width = 0.9 * getOption("width"), ellipsis = "[...]")
```

**Arguments**

- `str` *(character())*  
  Vector of strings.
- `width` *(integer(1))*  
  Width of the output.
- `ellipsis` *(character(1))*  
  If the string has to be shortened, this is signaled by appending `ellipsis` to `str`. Default is "[...]".

**Value**

*(character()).*

**Examples**

```r
str_trunc("This is a quite long string", 20)
```
to_decimal

Convert a Vector of Bits to a Decimal Number

Description

Converts a logical vector from binary to decimal. The bit vector may have any length, the last position is the least significant, i.e. bits are multiplied with \(2^{n-1}, 2^{n-2}, ..., 2^1, 2^0\) where \(n\) is the length of the bit vector.
Usage

to_decimal(bits)

Arguments

bits (logical())
Logical vector of input values. Missing values are treated as being FALSE. If bits is longer than 30 elements, an exception is raised.

Value

(integer(1)).

transpose_list

Transpose lists of lists

Description

Transposes a list of list, and turns it inside out, similar to the function transpose() in package purrr.

Usage

transpose_list(.l)

Arguments

.l (list() of list()).

Value

list().

Examples

```r
x = list(list(a = 2, b = 3), list(a = 5, b = 10))
str(x)
str(transpose_list(x))
```

```
# list of data frame rows:
transpose_list(iris[1:2, ])
```
### Unnest

**Unnest List Columns**

**Description**

Transforms list columns to separate columns, possibly by reference. The original columns are removed from the returned table. All non-atomic objects in the list columns are expanded to new list columns.

**Usage**

\[ \text{unnest}(x, \text{cols}, \text{prefix} = \text{NULL}) \]

**Arguments**

- **x** *(data.table::data.table())*
  - Data.table::data.table() with columns to unnest.
- **cols** *(character())*
  - Column names of list columns to operate on.
- **prefix** *(logical(1) | character(1))*
  - String to prefix the new column names with. Use \"{col}\" (without the quotes) as placeholder for the original column name.

**Value**

*(data.table::data.table()).*

**Examples**

```r
x = data.table::data.table(
  id = 1:2,
  value = list(list(a = 1, b = 2), list(a = 2, b = 2))
)
print(x)
unnest(data.table::copy(x), "value")
unnest(data.table::copy(x), "value", prefix = "\{col\}.")
```

### Which Min

**Index of the Minimum/Maximum Value, with Correction for Ties**

**Description**

Works similar to `base::which.min()`/`base::which.max()`, but corrects for ties. Missing values are treated as \( \text{Inf} \) for which_min and as \( -\text{Inf} \) for which_max().
with_package

Usage

which_min(x, ties_method = "random", na_rm = FALSE)

which_max(x, ties_method = "random", na_rm = FALSE)

Arguments

x (numeric())
  Numeric vector.

ties_method (character(1))
  Handling of ties. One of "first", "last" or "random" (default) to return the first
  index, the last index, or a random index of the minimum/maximum values.

na_rm (logical(1))
  Remove NAs before computation?

Value

(integer()): Index of the minimum/maximum value. Returns an empty integer vector for empty
input vectors and vectors with no non-missing values (if na_rm is TRUE). Returns NA if na_rm is
FALSE and at least one NA is found in x.

Examples

x = c(2, 3, 1, 3, 5, 1, 1)
which_min(x, ties_method = "first")
which_min(x, ties_method = "last")
which_min(x, ties_method = "random")

which_max(x)
which_max(integer(0))
which_max(NA)
which_max(c(NA, 1))

with_package

Execute code with a modified search path

Description

Attaches a package to the search path (if not already attached), executes code and eventually re-
moves the package from the search path again, restoring the previous state.

Note that this function is deprecated in favor of the (now fixed) version in withr.

Usage

with_package(package, code, ...)
Arguments

package (character(1))
Name of the package to attach.

code (expression)
Code to run.

... (any)
Additional arguments passed to library().

Value

Result of the evaluation of code.

See Also

withr package.

---

%nin%  Negated in-operator

Description

This operator is equivalent to !(x %in% y).

Usage

x %nin% y

Arguments

x (vector())
Values that should not be in y.

y (vector())
Values to match against.
Index

* Dictionary
  Dictionary, 15

* Internal
  rd_info, 38
  %nin%, 52

as_factor, 4
as_short_string, 5
assert_ro_binding, 4

base::cat(), 36, 37
base::cbind(), 9, 37
base::match(), 32
base::message(), 36
base::names(), 35
base::print(), 5
base::rbind(), 9
base::rep_len(), 32
base::requireNamespace(), 41
base::RNGkind(), 26
base::sample.int(), 45
base::seq_along(), 42
base::seq_len(), 42
base::split(), 8
base::sprintf(), 5, 37
base::stop(), 36
base::strwrap(), 37
base::warning(), 36
base::which.max(), 50
base::which.min(), 50
bibentry(), 24

calculate_hash, 6
callr::r(), 21
capitalize, 6
catf(printf), 36
check_packages_installed, 7
chunk(chunk_vector), 8
chunk_vector, 8
cite_bib(format_bib), 24
compat-map, 9
compose, 11
compute_mode, 12
count_missing, 13
crate, 13
cross_join, 14
data.table(), 14
data.table::CJ(), 14
data.table::data.table(), 6, 9, 15, 22, 23, 28, 37, 42, 48, 50
data.table::rbindlist(), 11
data.table::setkeyv(), 42
data.tables, 37
deframe (enframe), 22
detect (compat-map), 9
Dictionary, 15, 15, 18
dictionary_sugar
  (dictionary_sugar_get), 18
dictionary_sugar_get, 18
dictionary_sugar_mget
  (dictionary_sugar_get), 18
did_you_mean, 19
digest::digest(), 6
discard (compat-map), 9
distinct_values, 20
do.call(), 29
encapsulate, 21
enframe, 22
every (compat-map), 9
extract_vars, 23

factor(), 4, 20
find.package(), 7
format_bib, 24
formula(), 23
formulate, 25
function(), 6
get_private, 25
get_seed, 26
has_element, 26
ids, 27
imap (compat-map), 9
imap_chr (compat-map), 9
imap_dbl (compat-map), 9
imap_dtc (compat-map), 9
imap_dtr (compat-map), 9
imap_int (compat-map), 9
imap_lgl (compat-map), 9
insert_named, 28
invoke, 29
is_scalar_na, 30
iwalk (compat-map), 9
keep (compat-map), 9
keep_in_bounds, 30
leanify_package (leanify_r6), 31
leanify_package(), 31
library(), 52
load_dataset, 32
logical(), 20
map (compat-map), 9
map_at (compat-map), 9
map_bc (compat-map), 9
map_br (compat-map), 9
map_chr (compat-map), 9
map_dbl (compat-map), 9
map_dtc (compat-map), 9
map_dtr (compat-map), 9
map_if (compat-map), 9
map_int (compat-map), 9
map_lgl (compat-map), 9
map_values, 32
messagef (printf), 36
mlr3misc (mlr3misc-package), 3
mlr3misc-package, 3
modify_at (modify_if), 33
modify_if, 33
named_list, 34
named_vector, 34
names2, 35
open_help, 36
ordered(), 20
paradox::ParamSet, 18
pmap (compat-map), 9
pmap_chr (compat-map), 9
pmap_dbl (compat-map), 9
pmap_dtc (compat-map), 9
pmap_dtr (compat-map), 9
pmap_int (compat-map), 9
pmap_lgl (compat-map), 9
printf, 36
proc.time(), 22
pwalk (compat-map), 9
R6::R6, 15, 31
R6::R6Class, 4, 25, 31
R6::R6Class(), 19
rcbind, 37
rd_format_packages (rd_info), 38
rd_format_range (rd_info), 38
rd_format_string (rd_info), 38
rd_info, 38
recycle_vectors, 39
register_namespace_callback, 39
remove_named (insert_named), 28
reorder_vector, 40
rep(), 39
require_namespaces, 41
rowwise_table, 42
seq_along0 (sequence_helpers), 42
seq_col (sequence_helpers), 42
seq_len0 (sequence_helpers), 42
seq_row (sequence_helpers), 42
sequence_helpers, 42
set_class, 43
set_col_names (set_names), 44
set_names, 44
setTimeLimit(), 21, 29
shuffle, 45
some (compat-map), 9
stats::formula(), 25
stopf (printf), 36
str collapse, 45
str collapse(), 46
str indent, 46
str trunc, 47
strwrap(), 46
to_decimal, 48
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>topo_sort</td>
<td>48</td>
</tr>
<tr>
<td>toRd()</td>
<td>24</td>
</tr>
<tr>
<td>traceback()</td>
<td>21</td>
</tr>
<tr>
<td>transpose_list</td>
<td>49</td>
</tr>
<tr>
<td>unnest</td>
<td>50</td>
</tr>
<tr>
<td>utils::adist()</td>
<td>19</td>
</tr>
<tr>
<td>utils::head()</td>
<td>46</td>
</tr>
<tr>
<td>vector()</td>
<td>13</td>
</tr>
<tr>
<td>walk(compat-map)</td>
<td>9</td>
</tr>
<tr>
<td>warningf (printf)</td>
<td>36</td>
</tr>
<tr>
<td>which_max (which_min)</td>
<td>50</td>
</tr>
<tr>
<td>which_min</td>
<td>50</td>
</tr>
<tr>
<td>with_package</td>
<td>51</td>
</tr>
</tbody>
</table>