

Package ‘zeallot’

May 27, 2025

Type Package

Title Multiple, Unpacking, and Destructuring Assignment

Version 0.2.0

Description Provides a `%<-%` operator to perform multiple, unpacking, and destructuring assignment in R. The operator unpacks the right-hand side of an assignment into multiple values and assigns these values to variables on the left-hand side of the assignment.

URL <https://github.com/r-lib/zeallot>

BugReports <https://github.com/r-lib/zeallot/issues>

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 3.2)

Suggests codetools, testthat, knitr, rmarkdown, purrr

NeedsCompilation no

Author Nathan Teetor [aut, cre],
Paul Teetor [ctb]

Maintainer Nathan Teetor <nate@haufin.ch>

Repository CRAN

Date/Publication 2025-05-27 10:00:02 UTC

Contents

destructure	2
operator	3
zeallot	5
zealous	6

Index	7
--------------	----------

destructure	<i>Destructure an object</i>
-------------	------------------------------

Description

destructure is used during unpacking assignment to coerce an object into a list. Individual elements of the list are assigned to names on the left-hand side of the unpacking assignment expression.

Usage

```
destructure(x)

## S3 method for class 'data.frame'
destructure(x)

## S3 method for class 'summary.lm'
destructure(x)

## Default S3 method:
destructure(x)
```

Arguments

x An R object.

Details

New implementations of destructure can be very simple. A new destructure implementation might only strip away the class of a custom object and return the underlying list structure. Alternatively, an object might destructure into a nested set of values and may require a more complicated implementation. In either case, new implementations must return a list object so %<-% can handle the returned value(s).

See Also

[%<-%](#)

Examples

```
# Data frames become a list of columns
destructure(faithful)

# A simple shape class
shape <- function(sides = 4, color = "red") {
  structure(
    list(sides = sides, color = color),
    class = "shape"
  )
}
```

```

}

## Not run:
# Cannot destructure the shape object _yet_
c(sides, color) %<-% shape()

## End(Not run)

# Implement a new destructure method for the shape class
destructure.shape <- function(x) {
  unclass(x)
}

# Now we can destructure shape objects
c(sides, color) %<-% shape()

c(sides, color) %<-% shape(3, "green")

```

operator

Multiple assignment operators

Description

Assign values to name(s).

Usage

```
x %<-% value
```

```
value %>-% x
```

Arguments

x	A name structure, see section below.
value	A list of values, vector of values, or R object to assign.

Value

%<-% and %>-% invisibly return value.

These operators are used primarily for their assignment side-effect. %<-% and %>-% assign into the environment in which they are evaluated.

Name Structure

The basics

At its simplest, the name structure is a single variable name, in which case `%<-%` and `%->%` perform regular assignment, `x %<-% list(1, 2, 3)` or `list(1, 2, 3) %->% x`.

To specify multiple variable names use `c()`, for example `c(x, y, z) %<-% c(1, 2, 3)`.

When value is neither an atomic vector nor a list, `%<-%` and `%->%` will try to destructure value into a list before assigning variables, see [destructure\(\)](#).

In-place assignment

One may also assign into a list or environment, `c(x, x[[1]]) %<-% list(list(), 1)`.

Nested names

One can also nest calls to `c()`, `c(x, c(y, z))`. This nested structure is used to unpack nested values, `c(x, c(y, z)) %<-% list(1, list(2, 3))`.

Collector variables

To gather extra values from the beginning, middle, or end of value use a collector variable. Collector variables are indicated with the `..` prefix, `c(..x, y) %<-% list(1, 2, 3, 4)`.

Skipping values

Use `.` in place of a variable name to skip a value without raising an error or assigning the value, `c(x, ., z) %<-% list(1, 2, 3)`.

Use `..` to skip multiple values without raising an error or assigning the values, `c(w, .., z) %<-% list(1, NA, NA, 4)`.

Default values

Use `=` with a value to specify a default value for a variable, `c(x, y = NULL) %<-% list(1)`.

Unfortunately, using a default value with in-place assignment raises an error because of R's syntax, `c(x, x[[1]] = 1) %<-% list(list())`.

Named assignment

Use `=` *without* a value to assign values by name, `c(two=) %<-% list(one = 1, two = 2, three = 3)`.

See Also

For more on unpacking custom objects please refer to [destructure\(\)](#).

Examples

```
# Basic usage
c(x, y) %<-% list(0, 1)

# Unpack and assign nested values
c(c(x, y), z) %<-% list(list(2, 3), list(3, 4))

# Assign columns of data frame
c(eruptions, waiting) %<-% faithful

# Assign specific columns by name
```

```

c(mpg=, hp=, gear=) %<-% mtcars

# Alternatively, assign a column by position
c(first_col, .., last_col) %<-% mtcars

# Skip initial values, assign final value
todo_list <- list("1. Make food", "2. Pack lunch", "3. Save world")

c(.., final_todo) %<-% todo_list

# Assign first name, skip middle initial, assign last name
c(first_name, ., last_name) %<-% c("Ursula", "K", "Le Guin")

# Unpack nested values w/ nested names
fibs <- list(1, list(2, list(3, list(5))))

c(f2, c(f3, c(f4, c(f5)))) %<-% fibs

# Unpack first numeric, leave rest
c(f2, ..rest) %<-% unlist(fibs)

# Swap values without using temporary variables
c(a, b) %<-% c("eh", "bee")

c(a, b) %<-% c(b, a)

# Handle missing values with default values
parse_time <- function(x) {
  strsplit(x, " ")[[1]]
}

c(hour, period = NA) %<-% parse_time("10:00 AM")

c(hour, period = NA) %<-% parse_time("15:00")

# Right operator
list(1, 2, "a", "b", "c") %>-% c(x, y, ..z)

```

zeallot

Multiple, unpacking, and destructuring assignment in R

Description

zeallot provides a `%<-%` operator to perform multiple assignment in R. To get started with zeallot be sure to read over the introductory vignette on unpacking assignment, `vignette('unpacking-assignment')`.

Author(s)

Maintainer: Nathan Teetor <nate@haufin.ch>

Other contributors:

- Paul Teetor [contributor]

See Also

[%<-%](#)

zealous

Allow zealous assignment

Description

Using zeallot within an R package may cause R CMD check to raise NOTEs concerning variables with "no visible binding". To avoid these NOTEs, include a call to zealous() in a package's .onLoad() function.

Usage

```
zealous()
```

Details

The R CMD check process uses a package {codetools} to check for assigned variables. However, due to the non-standard nature of zeallot assignment the codetools package does not identify these variables. To work around this, the zealous() function modifies the variables found by codetools to avoid the NOTEs raised by R CMD check.

Examples

```
.onLoad <- function(libname, pkgname) {  
  zealous()  
}
```

Index

`%->%` (operator), [3](#)

`%<-%` (operator), [3](#)

`destructure`, [2](#)

`destructure()`, [4](#)

operator, [3](#)

`zeallot`, [5](#)

`zeallot-package` (`zeallot`), [5](#)

`zeallous`, [6](#)