Package ‘zeallot’

October 14, 2022

Type Package
Title Multiple, Unpacking, and Destructuring Assignment
Version 0.1.0
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/nteetor/zeallot
BugReports https://github.com/nteetor/zeallot/issues
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 6.0.1
VignetteBuilder knitr
Suggests testthat, knitr, rmarkdown, purrr, magrittr
NeedsCompilation no
Author Nathan Teetor [aut, cre],
Paul Teetor [ctb]
Maintainer Nathan Teetor <nathanteetor@gmail.com>
Repository CRAN
Date/Publication 2018-01-28 16:14:13 UTC

R topics documented:

  destructure ................................................................. 2
  operator ........................................................................ 3
  zeallot ......................................................................... 7

Index 8
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)

Arguments

x  An R object.

Details

If x is atomic destructure expects length(x) to be 1. If a vector with length greater than 1 is passed to destructure an error is raised.

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(
  data.frame(x = 0:4, y = 5:9)
)

# strings are split into list of characters
destructure("abcdef")

# dates become list of year, month, and day
destructure(Sys.Date())

# create a new destructure implementation
shape <- function(sides = 4, color = "red") {
  structure(  
    list(sides = sides, color = color),  
    class = "shape"
  )
}
```r
## Not run:
# cannot destructure the shape object yet
c(sides, color) %<-% shape()

## End(Not run)

# implement `destructure` for shapes
destructure.shape <- function(x) {
  list(x$sides, x$color)
}

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

sides # 4
color # "red"

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

sides # 3
color # 'green'
```

---

**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 may be 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 a call to `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()`.

object parts
Like assigning a variable, one may also assign part of an object, `c(x, x[[1]]) %<-% list(list(), 1)`.

nested names
One can also nest calls to `c()` when needed, `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 a `...` prefix, `c(...start, z) %<-% 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 `=` to specify a default value for a variable, `c(x, y = NULL) %<-% tail(1, 2)`.

When assigning part of an object a default value may not be specified because of the syntax enforced by R. The following would raise an "unexpected '=' ...", `c(x, x[[1]] = 1) %<-% list(list())`.

See Also
For more on unpacking custom objects please refer to `destructure()`.

Examples

```r
# basic usage
c(a, b) %<-% list(0, 1)

a # 0
b # 1

# unpack and assign nested values
c(c(e, f), c(g, h)) %<-% list(list(2, 3), list(3, 4))

e # 2
f # 3
```
g # 4
h # 5

# can assign more than 2 values at once
c(j, k, l) %<-% list(6, 7, 8)

# assign columns of data frame
c(erupts, wait) %<-% faithful
erupts # 3.600 1.800 3.333 ..
wait # 79 54 74 ..

# assign only specific columns, skip
# other columns
c(mpg, cyl, disp, ...) %<-% mtcars
mpg # 21.0 21.0 22.8 ..
cyl # 6 6 4 ..
disp # 160.0 160.0 108.0 ..

# skip initial values, assign final value
TODOs <- list("make food", "pack lunch", "save world")
c(..., task) %<-% TODOs
task # "save world"

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

first # "Ursula"
last # "Le Guin"

# simple model and summary
mod <- lm(hp ~ gear, data = mtcars)

# extract call and fstatistic from
# the summary
c(modcall, ..., modstat, .) %<-% summary(mod)

modcall
modstat

# unpack nested values w/ nested names
fibs <- list(1, list(2, list(3, list(5))))
c(f2, c(f3, c(f4, c(f5)))) %<-% fibs
f2 # 1
f3 # 2
f4 # 3
f5 # 5
# unpack first numeric, leave rest
\[
c(f2, fibcdr) \leftarrow fibs
\]
\[
f2 \quad # 1
fibcdr \quad # list(2, list(3, list(5)))
\]

# swap values without using temporary variables
\[
c(a, b) \leftarrow c("eh", "bee")
\]
\[
a \quad # "eh"
b \quad # "bee"
\]
\[
c(a, b) \leftarrow c(b, a)
\]
\[
a \quad # "bee"
b \quad # "eh"
\]

# unpack `strsplit` return value
\[
names \leftarrow c("Nathan, Maria, Matt, Polly", "Smith, Peterson, Williams, Jones")
\]
\[
c(firsts, lasts) \leftarrow strsplit(names, ",")
\]
\[
firsts \quad # c("Nathan", "Maria", ..
lasts \quad # c("Smith", "Peterson", ..
\]

# handle missing values with default values
\[
parse_time <- function(x) {
  strsplit(x, " ")[[1]]
}
\]
\[
c(hour, period = NA) \leftarrow parse_time("10:00 AM")
\]
\[
hour \quad # "10:00"
period \quad # "AM"
\]
\[
c(hour, period = NA) \leftarrow parse_time("15:00")
\]
\[
hour \quad # "15:00"
period \quad # NA
\]

# right operator
\[
list(1, 2, "a", "b", "c") \rightarrow c(x, y, ..chars)
\]
\[
x \quad # 1
y \quad # 2
chars \quad # list("a", "b", "c")
\]

# magrittr chains, install.packages("magrittr") for this example
\[
if (requireNamespace("magrittr", quietly = TRUE)) {
  library(magrittr)
\]
\[
c("hello", "world!")
\]
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 <nathanteetor@gmail.com>

Other contributors:

• Paul Teetor [contributor]

See Also

%<-%
Index

%->% (operator), 3
%<-% (operator), 3

destructure, 2
destructure(), 4

operator, 3

zeallot, 7
zeallot-package (zeallot), 7