Package 'zeallot'

January 27, 2017

Type Package

Title Multiple and Unpacking Variable Assignment

Version 0.0.2

Description Provides a %<-% operator to perform multiple or unpacking assignment in R. The operator unpacks a list of values and assigns these values to a single or multiple corresponding names.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 5.0.1

VignetteBuilder knitr

Suggests testthat, knitr, rmarkdown, purrr

NeedsCompilation no

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

Maintainer Nathan Teetor <nathanteetor@gmail.com>

Repository CRAN

Date/Publication 2017-01-27 18:47:52

R topics documented:

`destructure` *De-structure an Object*

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` function might only strip away the class of a custom object and return the underlying list structure. Alternatively, an object might de-structure 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
df <- data.frame(x = 0:4, y = 5:9)

destructure(df)

# strings are split into a list of
# individual characters
destructure('abcdef')

# dates are destructureed into a 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'
  )
}

## Not run:
# cannot destructure the shape object yet
{sides: color} %<-% shape()

## End(Not run)

# implement a new destructure function
destructure.shape <- function(x) {
  list(x$sides, x$color)
}

# now we can destructure shape objects
{sides: color} %<-% destructure(shape())
sides  # 4
color  # 'red'

{sides: color} %<-% destructure(shape(3, 'green'))
sides  # 3
color  # 'green'
```

`zeallot` *Multiple and Unpacking Assignment in R*

Description

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

See Also

`%<-%`

`%<-%` *zeallot Assignment Operator*

Description

Assign values to name(s).

Usage

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

Arguments

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

Details

variable names

To separate variable names use colons, `a: b: c`.

To nest variable names use braces, `{a: {b: c}}`.

values

To unpack a vector of variables do not include braces, `a: b %<-% c(1,2)`.

Include braces to unpack a list of values, `{a: b} %<-% list(1,2)`.

When `value` is neither a vector nor a list, the zeallot operator will try to de-structure `value` into a list, see `destructure`.

Nesting names will unpack nested values, `{a: {b: c}} %<-% 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.

Collect starting values, `{...a: b: c} %<-% list(1, 2, 3, 4)`

Collect middle values, `{a: ...b: c} %<-% list(1, 2, 3, 4)`

Collect ending values, `{a: b: ...c} %<-% list(1, 2, 3, 4)`

skipping values

Use a period `.` in place of a variable name to skip a value without raising an error, `{a: .: c} %<-% list(1, 2, 3)`. Values will not be assigned to `.`.

Skip multiple values by combining the collector prefix and a period, `{a: ....: e} %<-% list(1, NA, NA, NA, 5)`.

Value

`%<-%` invisibly returns `value`.

`%<-%` is used primarily for its assignment side-effect. `%<-%` assigns into the environment in which it is evaluated.

See Also

`destructure`

Examples

```
# basic usage
{a: b} %<-% list(0, 1)

a  # 0
b  # 1
```

```
# no braces when unpacking vectors
c: d  %<-% c(0, 1)

c  # 0
d  # 1

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

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

# can assign more than 2 values
{j: k: l} %<-% list(6, 7, 8)

# assign columns of data frame
{num_erupts: till_next} %<-% faithful

num_erupts  # 3.600 1.800 3.333 ..
till_next   # 79 54 74 ..

# assign only specific columns, skip
# other columns
{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')

{....: task} %<-% TODOs

task  # 'save world'

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

first  # 'Ursula'
last   # 'Le Guin'

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

# extract call and fstatistic from
# the summary
{fcall: ....: ffstat: .} %<-% fsum
```

```
fcall
ffstat

# unpack nested values with
# nested names
fibs <- list(1, list(2, list(3, list(5))))

{f2: {f3: {f4: {f5}}}} %<-% fibs

f2  # 1
f3  # 2
f4  # 3
f5  # list(5) *!!*

# unpack first value (a numeric) and
# second value (a list)
{f2: fcdr} %<-% fibs

f2    # 1
fcdr  # list(2, list(3, list(5)))

# swap values without using a
# temporary variable
a: b %<-% c('eh', 'bee')
a  # 'eh'
b  # 'bee'

a: b %<-% c(b, a)
a  # 'bee'
b  # 'eh'

# unpack strsplit return value
names <- c('Nathan,Maria,Matt,Polly', 'Smith,Peterson,Williams,Jones')

{firsts: lasts} %<-% strsplit(names, ',')

firsts  # c('Nathan', 'Maria', ..
lasts   # c('Smith', 'Peterson', ..
```

Index