Package: listcompr (via r-universe)

September 12, 2024

Version 0.4.0

Date 2021-10-02

Title List Comprehension for R

Author Patrick Roocks <mail@p-roocks.de>

Maintainer Patrick Roocks <mail@p-roocks.de>

Description Syntactic shortcuts for creating synthetic lists, vectors, data frames, and matrices using list comprehension.

URL https://github.com/patrickroocks/listcompr

Depends R (>= 3.1.2)

License GPL (>= 2)

Suggests testthat, rmarkdown, knitr, dplyr (>= 1.0.0)

Collate 'listcompr.r' 'gen-list.r' 'eval.r' 'expand.r'

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

Repository https://patrickroocks.r-universe.dev

RemoteUrl https://github.com/patrickroocks/listcompr

RemoteRef HEAD

RemoteSha e53839a5bf256161cd5adff7761edb079de3dbc1

Contents

gen.list 2
gen.list.expr 5
gen.logical.and 6
gen.named.list 7
listcompr 8

Index **10**

`gen.list` *Generate Lists, Vectors, Data Frames and Matrices with List Comprehension*

Description

Functions to transform a base expression containing free variables into a list, a vector, a data frame, or a matrix based on variable ranges and additional conditions.

Usage

```
gen.list(.expr, ...)

gen.vector(.expr, ...)

gen.data.frame(.expr, ..., byrow = FALSE)

gen.matrix(.expr, ..., byrow = FALSE)
```

Arguments

`.expr`	A base expression containing free variables which is evaluated for all combinations of variables, where the combinations of variables are given by the ranges and conditions (see `...` parameters). Expected structure of `.expr`: • For `gen.list` it may have arbitrary structure (including a list). • For `gen.vector` a value (i.e., a vector of length 1) is expected. • For `gen.data.frame` a (named) vector or list is expected which describes exactly one row of the data frame. Use `list(name = val)` if `val` is a non-fundamental type like `difftime`. • For `gen.matrix` either a (named) vector/list (like `gen.data.frame`) or a scalar is expected. In the first case, we expect the same as for `gen.data.frame`. In the latter case we expect exactly two variables (inducing rows and columns where the order depends on `byrow`) within the `...` arguments. Within `.expr` it is allowed to use functions and predefined constants from the parent environment.
`...`	Arbitrary many variable ranges and conditions. For all free variables occurring in `.expr` a range must be assigned, e.g., `x = 1:3, y = 1:5` for an expression `x + y`. At least one variable range is required. The ranges may depend on each other, e.g., `x = 1:3, y = x:3` or a substitution like `x = 1:3, y = 2 * x` is allowed. The generated values can be further restricted by conditions like `x <= y`.
`byrow`	Logical. If `FALSE` (the default), the elements of a vector within `.expr` are taken as columns. Otherwise, they are taken as rows.

Value

The result of `gen.list` is a list (a vector for `gen.vector`) containing an entry for each combination of the free variables (i.e., the Cartesian product), where all the free variables in `.expr` are substituted. The function `gen.vector` returns a vector while `gen.list` may contain also more complex substructures (like vectors or lists).

The output of `gen.data.frame` is a data frame where each substituted `.expr` entry is one row. The base expression `.expr` should contain a (named) vector or list, such that each entry of this vector becomes a column of the returned data frame. If the vector contains a single literal without a name, this is taken as column name. For instance, `gen.data.frame(a, a = 1:5)` returns the same as `gen.data.frame(c(a = a), a = 1:5)`. Default names 'V1', 'V2', ... are used, if no names are given and names can't be automatically detected.

The result of `gen.matrix`:

- It's similar to `gen.data.frame`, if `.expr` evaluates to a vector of length > 1, or row/column names are given. Each substituted `.expr` entry is one row of the matrix. In contrast to `gen.data.frame`, column names are not auto-generated, e.g., `gen.matrix(c(a_1, a_2), a_ = 1:2)` is an unnamed matrix. If the `.expr` argument has explicit names (e.g., `c(a_1 = a_1, a_2 = a_2)`), these column names are assigned to the resulting matrix.
- It's a matrix where the rows and columns are induced by the two variables within `...`, if `.expr` is a scalar, and no names or conditions are given. If `byrow` is `FALSE`, the second variable (i.e., the inner loop) refers to the columns, otherwise it refers to the rows. For instance, `gen.matrix(i + j, i = 1:3, j = 1:2)` is a matrix with 3 rows and 2 columns. For `gen.matrix(i + j, i = 1:3, j = 1:2, byrow = TRUE)` we get 2 rows and 3 columns.

All expressions and conditions are applied to each combination of the free variables separately, i.e., they are applied row-wise and not vector-wise. For instance, the term `sum(x,y)` (within `.expr` or a condition) is equivalent to `x+y`.

Indices for variables

A range for a variable ending with an underscore (like `x_`) defines a set of ranges affecting all variables named `{varname}_{index}`, e.g. `x_1`. For instance, in `gen.vector(x_1 + x_2 + x_3, x_ = 1:5)` the variables `x_1`, `x_2`, `x_3` are all ranging in `1:5`. This can be overwritten for each single `x_i`, e.g., an additional argument `x_3 = 1:3` assigns the range `1:3` to `x_3` while `x_1` and `x_2` keep the range `1:5`. A group of indexed variables is kept always sorted according to the position of the main variable `{varname}_`. For instance, the two following statements produce the same results:

- `gen.vector(x_1 + x_2 + a, x_ = 1:5, a = 1:2, x_1 = 1:2)`
- `gen.vector(x_1 + x_2 + a, x_1 = 1:2, x_2 = 1:5, a = 1:2)`

Folded expressions

Expressions and conditions support a `...`-notation which works as follows:

- A vector like `c(x_1, ..., x_4)` is a shortcut for `c(x_1, x_2, x_3, x_4)`.
- A named vector like `c(a_1 = x_1, ..., a_3 = x_3)` is a shortcut for `c(a_1 = x_1, a_2 = x_2, a_3 = x_3)`.

- A n-ary function argument like `sum(x_1, ..., x_4)` is a shortcut for `sum(x_1, x_2, x_3, x_4)`.
- Repeated expressions of binary operators can be abbreviated with the `...` expressions as follows: `x_1 + ... + x_4` is a shortcut for `x_1 + x_2 + x_3 + x_4`. Note that, due to operator precedence, `1 + x_1 + ... + x_4` will not work, but `1 + (x_1 + ... + x_4)` works as expected.
- For non-commutative operators, `x_1 - ... - x_4` is a shortcut for `x_1 - x_2 - x_3 - x_4` which is evaluated as `((x_1 - x_2) - x_3) - x_4`.

The conditions may contain itself list comprehension expressions, e.g., `gen.logical.and` to compose and-connected logical expressions.

Character patterns

In expression there may occur characters with `{}`-placeholders. The content of these placeholders is evaluated like any other part of an expression and converted to a character. For example, `"a{x}"` is transformed into `"a1"` for `x = 1`. Double brackets are transformed into a single bracket without evaluating the inner expression. For instance, `"var{x + 1}_{{a}}"` is transformed into `"var2_{a}"` for `x = 1`.

See Also

`gen.named.list` to generate named structures, `gen.list.expr` to generate expressions to be evaluated later, `gen.logical.and` to generate logical and/or conditions, and listcompr for an overview of all list comprehension functions.

Examples

```
# Sum of 1:x
gen.vector(sum(1:x), x = 1:10)

# Same as above, but return as text
gen.list("sum of 1 to {x} is {sum(1:x)}", x = 1:5)

# A list containing vectors [1], [1, 2], [1, 2, 3], ...
gen.list(gen.vector(i, i = 1:n), n = 1:10)

# A data frame of tuples (x_1, x_2, x_3) summing up to 10
gen.data.frame(c(x_1, ..., x_3), x_ = 1:10, x_1 + ... + x_3 == 10)

# Same as above, but restrict to ascending tuples with x_i <= x_(i+1)
gen.data.frame(c(x_1, ..., x_3), x_1 = 1:10, x_2 = x_1:10, x_3 = x_2:10,
               x_1 + ... + x_3 == 10)

# A data frame containing the numbers in 2:20, the sum of their divisors
# and a flag if they are "perfect" (sum of divisors equals the number)
gen.data.frame(list(n, sumdiv, perfect = (n == sumdiv)), n = 2:20,
               sumdiv = sum(gen.vector(x, x = 1:(n-1), n %% x == 0)))

# A diagonal matrix with (1, ..., 5) on the diagonal
gen.matrix(if (i == j) i else 0, i = 1:5, j = 1:5)
```

gen.list.expr *Generate List and Vector Expressions with List Comprehension*

Description

Functions to transform a base expression containing free variables into a list or a vector of expressions, based on variable ranges and additional conditions.

Usage

```
gen.list.expr(.expr, ...)

gen.vector.expr(.expr, ...)

gen.named.list.expr(.str, .expr, ...)

gen.named.vector.expr(.str, .expr, ...)
```

Arguments

`.expr`	A base expression which is partially evaluated for all combinations of variables. It may still contain free variables.
`...`	Arbitrary many variable ranges and conditions.
`.str`	A character pattern, containing expressions to be evaluated in {}-brackets.

Details

See `gen.list` for more details on the `.expr` and `...` parameters.

See `gen.named.list` for more details on the `.str` parameter.

For variables with underscores additionally the evaluation of indices in `()`-brackets is supported. For example, an expression `x_(i+1)` is evaluated as `x_3` for `i = 2`.

Value

Returns an expression containing a list or a vector which might be evaluated later. The argument `.expr` is partially evaluated, where all free variables are substituted for which a range is given. The other variables remain untouched.

See Also

`gen.list` to generate lists, `gen.named.list` to generate named lists, and listcompr for an overview of all list comprehension functions.

Examples

```
# An expression which is partially evaluated
gen.list.expr(a_i + 2 * i, i = 1:4)

# Generate an expression with placeholders a_i,
# generate data for a_1, ..., a_4 and finally evaluate it
expr <- gen.vector.expr(a_i + a_(j+1), i = 1:3, j = 1:3, i != j)
data <- gen.data.frame(c(a_1 = a_1, ..., a_4 = a_4), a_ = 1:2)
eval(expr, data)
```

`gen.logical.and` *Generate Logical Conditions with List Comprehension*

Description

Functions to compose and-/or-connected logical conditions, based on variable ranges and additional conditions.

Usage

```
gen.logical.and(.expr, ...)

gen.logical.or(.expr, ...)
```

Arguments

`.expr`	A base expression which is partially evaluated for all combinations of variables. It may still contain free variables.
`...`	Arbitrary many variable ranges and conditions.

Details

See `gen.list` for more details on the `.expr` and `...` parameters.

For variables with underscores additionally the evaluation of indices in `()`-brackets is supported. For example, an expression `x_(i+1)` is evaluated as `x_3` for `i = 2`.

Value

Returns an expression `expr_1 & ... & expr_n` or `expr_1 | ... | expr_n` where `expr_i` is generated from `.expr`, where all free variables are substituted for which a range is given. The other variables remain untouched.

The generated condition may be used within the the conditions of `gen.list` and similar functions from this package.

See Also

gen.list to generate lists and thereby make use of the generated logical conditions, and listcompr for an overview of all list comprehension functions.

Examples

```
# Returns a_1 == 1 & a_2 == 2 & a_3 == 3
gen.logical.and(a_i == i, i = 1:3)

# A data frame of tuples (x_1, x_2, x_3, x_4) summing up to 10 with x_i <= x_(i+1)
gen.data.frame(c(x_1, ..., x_4), x_ = 1:10, x_1 + ... + x_4 == 10,
               gen.logical.and(x_i <= x_(i+1), i = 1:3))

# Get all permutations of 1:4
gen.data.frame(c(a_1, ..., a_4), a_ = 1:4,
               gen.logical.and(a_i != a_j, i = 1:4, j = (i+1):4))

# Get again the permutations of 1:4, using filter from dplyr
df <- gen.data.frame(c(a_1, ..., a_4), a_ = 1:4)
dplyr::filter(df, !!gen.logical.and(a_i != a_j, i = 1:3, j = (i+1):4))
```

---

gen.named.list *Generate Named Lists, Vectors, Data Frames, and Matrices with List Comprehension*

Description

Functions to transform patterns with placeholders into characters or into names of lists, vectors, data frames or matrices, based on variable ranges and additional conditions.

Usage

```
gen.named.list(.str, .expr, ...)

gen.named.vector(.str, .expr, ...)

gen.named.data.frame(.str, .expr, ..., byrow = FALSE)

gen.named.matrix(.str, .expr, ..., byrow = FALSE)
```

Arguments

.str	A character, containing expressions to be evaluated in {}-brackets, e.g., "a{x}" is transformed into "a1" for x = 1. Double brackets are transformed into a single bracket without evaluating the inner expression. For instance, "var{x + 1}_{{a}}" is transformed into "var2_{a}" for x = 1.

`.expr`	A base expression containing free variables which is evaluated for all combinations of variables.
`...`	Arbitrary many variable ranges and conditions.
`byrow`	Logical. If `FALSE` (the default), the elements of an `.expr` vector are taken as columns. Otherwise, they are taken as rows.

Details

The free variables in the inner expressions (i.e., the content of the `{}`-brackets) of `.expr` are evaluated in the same way as expressions in `gen.list`.

See `gen.list` for more details on the `.expr` and `...` parameters.

Value

These functions return lists, vectors, data frames, and matrices. They work very similar to their counterparts without ".named". Additionally the vector of characters, induced by `.str`, serves as a vector of names for the generated structures. In case of lists or vectors, the result is a named list or a named vector. For data frames and matrices, the names are taken as row names.

See Also

`gen.list` for explanations on list and vector comprehension, and listcompr for an overview of all list comprehension functions.

Examples

```
# sum up 1:i for i in 1:5
gen.named.list("sum_to_{x}", sum(1:x), x = 1:5)

# matrix with named columns and rows
gen.named.matrix("row{i}", gen.named.vector("col{j}", i+j, j = 1:3), i = 1:3)

# a matrix where the expression refers to the rows and not the columns
gen.named.matrix("col{i}", c(row1 = i, row2 = 10 * i, row3 = 100 * i), i = 1:10,
                 byrow = TRUE)
```

`listcompr` *Summary of the listcompr Package*

Description

The listcompr package offers some syntactic shortcuts to create lists, vectors and data frames containing values within a given range with given conditions. It is a light-weight package written in base R without any compiled code or dependencies to other packages.

Functions

- The main functionality of listcompr: generate lists, vectors, and data frames: `gen.list`
- Generate named lists, vectors, and data frames: `gen.named.list`
- Generate expressions containing lists and vectors: `gen.list.expr`
- Generate conditions to be used in other functions of listcompr: `gen.logical.and`

Vignettes

To learn the basics of listcompr, start with the vignette:

```
vignette("introduction", package = "listcompr")
```

Contact

To submit bugs, feature requests or other comments, feel free to write a mail to me.

Author(s)

Patrick Roocks, <mail@p-roocks.de>

Index

gen.data.frame *(*gen.list*)*, 2
gen.list, 2, *5–9*
gen.list.expr, *4*, 5, *9*
gen.logical.and, *4*, 6, *9*
gen.logical.or *(*gen.logical.and*)*, 6
gen.matrix *(*gen.list*)*, 2
gen.named.data.frame *(*gen.named.list*)*, 7
gen.named.list, *4, 5*, 7, *9*
gen.named.list.expr *(*gen.list.expr*)*, 5
gen.named.matrix *(*gen.named.list*)*, 7
gen.named.vector *(*gen.named.list*)*, 7
gen.named.vector.expr *(*gen.list.expr*)*, 5
gen.vector *(*gen.list*)*, 2
gen.vector.expr *(*gen.list.expr*)*, 5

listcompr, *4, 5*, *7, 8*, 8