Package ‘functools’

August 29, 2016

Title  Functional Programming in R
Version  0.2.0
Description  Extends functional programming in R by
providing support to the usual higher order functional
suspects (Map, Reduce, Filter, etc.).
URL  https://github.com/paulhendricks/functools
BugReports  https://github.com/paulhendricks/functools/issues
Depends  R (>= 3.1.2)
License  MIT + file LICENSE
LazyData  true
Suggests  testthat, memoise, pryr
NeedsCompilation  no
Author  Paul Hendricks [aut, cre]
Maintainer  Paul Hendricks <paul.hendricks.2013@owu.edu>
Repository  CRAN
Date/Publication  2015-09-02 16:55:15

R topics documented:

  All ................................................................. 2
  Always ......................................................... 3
  Andify .......................................................... 4
  Any ............................................................... 5
  Apply ............................................................ 6
  Best ............................................................. 6
  Compact ......................................................... 7
  Compose ......................................................... 8
  Existy .......................................................... 8
  Fail_With ....................................................... 9
  False ........................................................... 10
  functools ....................................................... 11
Identity ............................................................... 11
Lapply ................................................................. 12
Mapply ................................................................. 12
Memoise ............................................................... 13
Na ................................................................. 14
Null ................................................................. 14
Orify ................................................................. 15
Partial ............................................................... 15
Reduce_Right ......................................................... 16
Reject ............................................................... 16
Sapply ............................................................... 17
Tapply ............................................................... 18
True ................................................................. 18
Truthy .............................................................. 19
Vapply ............................................................... 20

Index 21

---

**All**

*Test if all items in an object evaluate to TRUE.*

---

**Description**

All() is a predicate functional that takes a predicate function .f and an iterable object .x and:

1. iterates over each item i in object .x,
2. evaluates .f(i),
3. and ultimately returns TRUE if all items i in object .x evaluate as TRUE.

**Usage**

```r
All(.x, .f, ..., na.rm = FALSE)
```

**Arguments**

- `.x` An iterable object.
- `.f` A predicate function.
- `...` Further arguments passed to the predicate function.
- `na.rm` A logical value indicating whether NA values should be stripped before the computation proceeds.

**Value**

A logical value indicating if all items evaluated as TRUE.
See Also

*Any* to test if all items in an object evaluate to TRUE.

Other predicate functionals: *Any; Reject*

Examples

```r
# Examples
data(mtcars)
all(mtcars, is.numeric) # TRUE
all(mtcars, is.character) # FALSE
mtcars$am <- factor(mtcars$am)
all(mtcars, is.numeric) # FALSE
all(mtcars, is.factor) # FALSE

# Handles NAs and NULLs
all(list(NA, "3", NULL), is.numeric) # FALSE
all(list(NA, 3, NULL), is.numeric) # FALSE
all(list(NA, "3", NULL, 5), is.numeric) # FALSE

# Use na.rm = TRUE to remove NAs and NULLS
all(list(NA, TRUE), Identity) # NA
all(list(NA, TRUE), Identity, na.rm = TRUE) # TRUE
```

Always

Create a function that always returns a specific object.

Description

*Always*(.x) is a closure function that takes any object .x, and returns a function that always returns object .x.

Usage

*Always*(.x)

Arguments

.x

An object.

Value

A function that itself returns .x.

Examples

```r
# comment here
always_0 <- *Always*(0)
always_0() # 0
always_true <- *Always*(TRUE)
always_true() # TRUE
```
Andify

Predicate function operator that creates new predicate functions linked by the && operator.

Description

Predicate function operator that creates new predicate functions linked by the && operator.

Usage

Andify(...)

Arguments

...  n functions to apply in order from left to right.

Value

A predicate function linked by the && operator.

See Also

Orify to create new predicate functions linked by the || operator.

Other predicate function operators: Orify

Examples

# Examples
is_numeric <- is.numeric
is_even <- function(x) x %% 2 == 0
greater_than_10 <- function(x) x > 10
less_than_100 <- function(x) x < 100
even_number_between_10_and_100 <-
    Andify(is_numeric, is_even, greater_than_10, less_than_100)
even_number_between_10_and_100(8) # FALSE
even_number_between_10_and_100(9) # FALSE
even_number_between_10_and_100(10) # FALSE
even_number_between_10_and_100(11) # FALSE
even_number_between_10_and_100(12) # TRUE
even_number_between_10_and_100(49) # FALSE
even_number_between_10_and_100(50) # TRUE
even_number_between_10_and_100(100) # FALSE
even_number_between_10_and_100(101) # FALSE
even_number_between_10_and_100(102) # FALSE
Any

Test if any items in an object evaluate to TRUE.

Description

Any() is a predicate functional that takes a predicate function \( f \) and an iterable object \( x \) and:

1. iterates over each item \( i \) in object \( x \),
2. evaluates \( f(i) \),
3. and ultimately returns TRUE if any items \( i \) in object \( x \) evaluate as TRUE.

Usage

\[
\text{Any}(.x, .f, \ldots, \text{na.rm} = \text{FALSE})
\]

Arguments

- \( .x \): An iterable object.
- \( .f \): A predicate function.
- \( \ldots \): Further arguments passed to the predicate function.
- \( \text{na.rm} \): A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

A logical value indicating if any items evaluated as TRUE.

See Also

All to test if all items in an object evaluate to TRUE.

Other predicate functionals: All; Reject

Examples

```r
# Examples
data(mtcars)
Any(mtcars, is.numeric) # TRUE
Any(mtcars, is.character) # FALSE
mtcars$am <- factor(mtcars$am)
Any(mtcars, is.numeric) # TRUE
Any(mtcars, is.factor) # TRUE

# Handles NAs and NULLs
Any(list(NA, "3", NULL), is.numeric) # FALSE
Any(list(NA, 3, NULL), is.numeric) # TRUE
Any(list(NA, "3", NULL, 5), is.numeric) # TRUE
```
Apply

Wrapper for apply function.

Description

Wrapper for apply.

Usage

Apply(.x, .f, .margin, ...)

Arguments

.x An array, including a matrix.
.f A function to be applied.
.margin A vector giving the subscripts which the function will be applied over.
... Optional arguments to f.

See Also

apply for code and documentation.

Other functionals: Lapply; Mapply; Sapply; Tapply; Vapply

Best

Find the best value in a vector.

Description

Best() takes a vector .x and a binary predicate function .f and returns the result of .f reduced over .x.

Usage

Best(.x, .f)

Arguments

.x A vector.
.f A binary predicate function.
Compact

Value

The best value in that vector, as determined by the binary predicate function.

Examples

# Simulate the behavior of max with numerics
Best(1:10, function(x, y) return(x > y))
# Simulate the behavior of min with numerics
Best(1:10, function(x, y) return(x < y))
# This comparison function prefers values that begin with l
Best(letters, function(x, y) return(x[1] == "l"))

Compact

Filter NA and NULL values out of a vector, list, or data.frame.

Description

Compact() takes a vector x and returns it with all NULL and NA values filtered out.

Usage

Compact(.x)

Arguments

.x 
A vector.

Value

Vector .x but with all NULL and NA values filtered out.

Examples

# Removes all null elements from a vector:
a <- list(NULL, 1, 5, NULL)
Compact(a)

b <- c(1, 2, 0, 4, NULL, 1, 3, NULL)
Compact(b)
Compose

Compose multiple functions.

Description

In infix and prefix forms.

Usage

Compose(...)

.f %O% .g

Arguments

... n functions to apply in order from right to left.
.f A function.
.g A function.

Value

A function that will apply each function in order from right to left.

See Also

Other function operators: Fail_With; Memoise; Partial; Reduce_Right

Examples

not_null <- `!` %O% is.null
not_null(4)
not_null(NULL)

add1 <- function(x) x + 1
Compose(add1,add1)(8)

Description

Existy() returns TRUE or FALSE if an object exists or not. An object exists if it is not NULL or NA.
Fail_With

Usage

Existy(.x)

Arguments

.x an object.

Value

a logical value.

See Also

Other predicate functions: Truthy

Examples

# Some examples
Existy(4) # TRUE
Existy("foo") # TRUE
Existy(NULL) # FALSE
Existy(NA) # FALSE

# Works with lists
Existy(list(4, "foo", NULL, NA)) # TRUE
Existy(list(4, "foo")) # TRUE
Existy(list(NULL)) # TRUE
Existy(list(NA)) # FALSE

# Works with applying over lists
lapply(list(4, "foo", NULL, NA), Existy) # TRUE, TRUE, FALSE, FALSE

Fail_With

Fail with a default value.

Description

Fail_With() turns a function that throws an error into a function that returns a default value when there is an error. The essence of Fail_With() is simple: it is just a wrapper around try(), the function that captures errors and allows execution to continue.

Usage

Fail_With(.default = NULL, .f, .silent = FALSE)
Arguments

- `.default` default value.
- `.f` any function that throws an error.
- `.silent` logical: should the report of error messages be suppressed?

Value

A function that returns a default value when there's an error.

See Also

Other function operators: Compose, `%>%`, Memoise, Partial, Reduce_Right

Description

False() is a function that returns FALSE.

Usage

False()

Value

FALSE.

See Also

Other constants: Identity, Na, Null, True

Examples

# False() returns FALSE:
False()
functools

functools: Extending Functional Programming in R

Description

functools extends functional programming in R. It has three main goals:

Details

• Add support to the usual higher order functional suspects (Map, Reduce, Filter, etc.) without extending any core R objects.
• Use a consistent API to access different functionals in base R such as ‘lapply’ or ‘apply’.
• Provide blazing fast performance for in-memory data by writing key pieces in C++ and options for parallelization, where possible.

functools achieves these goals through three main types of function design patterns:

• Closures (functions that take data and return functions)
• Functionals (functions that take functions and return data)
• Function Operators (functions that take functions and return functions)

To learn more about functools, start with the vignettes: browseVignettes(package = "functools")

Identity

Description

Identity() returns itself.

Usage

Identity(x)

Arguments

x an object.

Value

the object.

See Also

Other constants: False; Na; Null; True
Examples

# Return itself:
Identity(5)
Identity(mean)
Identity(lm(data = mtcars, mpg ~ cyl))

Lapply

Wrapper for lapply function.

Description

Wrapper for lapply.

Usage

Lapply(.x, .f, ...)

Arguments

.x A vector.
.f A function to be applied.
... Optional arguments to .f.

See Also

lapply for code and documentation.

Other functionals: Apply: Mapply; Sapply; Tapply; Vapply

Mapply

Wrapper for mapply function.

Description

Wrapper for mapply.

Usage

Mapply(..., .f, more_args = NULL, simplify = TRUE, use_names = TRUE)
Arguments

... Arguments to vectorize over (vectors or lists of strictly positive length, or all of zero length).
.f A function to be applied.
more_args A list of other arguments to FUN.
simplify Logical or character string; attempt to reduce the result to a vector, matrix or higher dimensional array; see the simplify argument of `sapply`.
use_names Logical; use names if the first ... argument has names, or if it is a character vector, use that character vector as the names.

See Also

`mapply` for code and documentation.

Other functionals: `Apply; Lapply; Sapply; Tapply; Vapply`

---

Memoise

Memoise a function.

Description

Wrapper for `memoise`.

Usage

Memoise(.f)

Arguments

.f Function of which to create a memoised copy.

Value

A memoised copy of the original function.

See Also

`memoise` for code and documentation.

Other function operators: `Compose, %O%; Fail_With; Partial; Reduce_Right`
### Description
Na() is a function that returns NA.

### Usage
Na()

### Value
NA.

### See Also
Other constants: False; Identity; Null; True

### Examples
# Na() returns NA:
Na()

---

### Description
Null() is a function that returns NULL.

### Usage
Null()

### Value
NULL.

### See Also
Other constants: False; Identity; Na; True

### Examples
# Null() returns NULL:
Null()
**Orify**

Predicate function operator that creates new predicate functions linked by the \( \| \) operator.

**Description**

Predicate function operator that creates new predicate functions linked by the \( \| \) operator.

**Usage**

`Orify(...)`

**Arguments**

... n functions to apply in order from left to right

**Value**

A predicate function linked by the \( \| \) operator.

**See Also**

`Andify` to create new predicate functions linked by the \&\& operator.

Other predicate function operators: `Andify`

**Examples**

```r
# Examples
is_character_or_factor <- Orify(is.character, is.factor)
is_character_or_factor(letters) # TRUE
is_character_or_factor(factor(state.abb)) # TRUE
is_character_or_factor(1:100) # FALSE
```

**Partial**

Partial apply a function, filling in some arguments.

**Description**

Wrapper for `partial`.

**Usage**

`Partial(...)`

**Arguments**

... Arguments to be passed to `partial`.
See Also

partial for code and documentation.

Other function operators: Compose, %O%; Fail_With; Memoise; Reduce_Right

---

**Reduce_Right**

*Simple wrapper for Reduce, proceeding from the right.*

**Description**

Wrapper for Reduce with right set to TRUE.

**Usage**

```r
Reduce_Right(...) 
```

**Arguments**

...  

Arguments to be passed to Reduce.

**See Also**

Reduce for code and documentation.

Other function operators: Compose, %O%; Fail_With; Memoise; Partial

---

**Reject**

*Reject*

**Description**

Reject() is the opposite of Filter. Reject applies the negation of the unary predicate function f to each element of x, coercing to logical if necessary, and returns the subset of x for which this gives true. Note that possible NA values are currently always taken as false; control over NA handling may be added in the future.

**Usage**

```r
Reject(f, x) 
```

**Arguments**

f  

a predicate function.

x  

a vector.
Sapply

Value
x filtered where f applies

See Also
Other predicate functionals: All; Any

Examples

# Some examples
Filter(function(x) x < 5, 1:10)
Reject(function(x) x < 5, 1:10)

Description
Wrapper for sapply.

Usage
Sapply(.x, .f, ..., simplify = TRUE, use_names = TRUE)

Arguments

.x A vector.
.f A function to be applied.
... Optional arguments to f.
simplify Logical or character string: should the result be simplified to a vector, matrix or higher dimensional array if possible?
use_names Logical; if TRUE and if .x is character, use .x as names for the result unless it had names already.

See Also
sapply for code and documentation.

Other functionals: Apply; Lapply; Mapply; Tapply; Vapply
Tapply

Wrapper for tapply function.

Description

Wrapper for tapply.

Usage

Tapply(.x, index, .f = NULL, ..., simplify = TRUE)

Arguments

.x
A vector.

index
List of one or more factors, each of same length as .x. The elements are coerced
to factors by as.factor.

.f
A function to be applied.

... Optional arguments to .f.

simplify
If FALSE, tapply always returns an array of mode "list". If TRUE (the default),
then if FUN always returns a scalar, tapply returns an array with the mode of the
scalar.

See Also

tapply for code and documentation.

Other functionals: Apply; Lapply; Mapply; Sapply; Vapply

True

Description

True() is a function that returns TRUE.

Usage

True()

Value

TRUE.

See Also

Other constants: False; Identity; Na; Null
Examples

# True() returns TRUE:
True()

Description

Truthy() returns TRUE or FALSE if an object is TRUE or not. An object is is "TRUE" if it is exists and is TRUE.

Usage

Truthy(.x)

Arguments

.x an object.

Value

a logical value.

See Also

Other predicate functions: Exist

Examples

# Returns if a value exists or not:
Truthy(TRUE) # TRUE
Truthy(FALSE) # FALSE
Truthy(NULL) # FALSE
Truthy(NA) # FALSE
Truthy(2L) # TRUE
Truthy(1L) # TRUE
Truthy(0L) # FALSE
Truthy("a") # TRUE
Vapply

Description

Wrapper for `vapply`.

Usage

Vapply(.x, .f, fun_value, ..., use_names = TRUE)

Arguments

- `.x` A vector.
- `.f` A function to be applied.
- `fun_value` A (generalized) vector; a template for the return value from `.f`.
- `...` Optional arguments to `.f`.
- `use_names` Logical; if TRUE and if X is character, use `.x` as names for the result unless it had names already.

See Also

- `vapply` for code and documentation.
- Other functionals: `Apply`; `Lapply`; `Mapply`; `Sapply`; `Tapply`
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>%0% (Compose)</td>
<td>8</td>
</tr>
<tr>
<td>%0%</td>
<td>10, 13, 16</td>
</tr>
<tr>
<td>All</td>
<td>2, 5, 17</td>
</tr>
<tr>
<td>Always</td>
<td>3</td>
</tr>
<tr>
<td>Andify</td>
<td>4, 15</td>
</tr>
<tr>
<td>Any</td>
<td>3, 5, 17</td>
</tr>
<tr>
<td>Apply</td>
<td>6, 12, 13, 17, 18, 20</td>
</tr>
<tr>
<td>apply</td>
<td>6</td>
</tr>
<tr>
<td>Best</td>
<td>6</td>
</tr>
<tr>
<td>Compact</td>
<td>7</td>
</tr>
<tr>
<td>Compose</td>
<td>8, 10, 13, 16</td>
</tr>
<tr>
<td>Existy</td>
<td>8, 19</td>
</tr>
<tr>
<td>Fail_With</td>
<td>8, 9, 13, 16</td>
</tr>
<tr>
<td>False</td>
<td>10, 11, 14, 18</td>
</tr>
<tr>
<td>functools</td>
<td>11</td>
</tr>
<tr>
<td>functools-package (functools)</td>
<td>11</td>
</tr>
<tr>
<td>Identity</td>
<td>10, 11, 14, 18</td>
</tr>
<tr>
<td>Lapply</td>
<td>6, 12, 13, 17, 18, 20</td>
</tr>
<tr>
<td>lapply</td>
<td>12</td>
</tr>
<tr>
<td>Maply</td>
<td>6, 12, 12, 17, 18, 20</td>
</tr>
<tr>
<td>mapply</td>
<td>12, 13</td>
</tr>
<tr>
<td>Memoise</td>
<td>8, 10, 13, 16</td>
</tr>
<tr>
<td>memoise</td>
<td>13</td>
</tr>
<tr>
<td>Na</td>
<td>10, 11, 14, 18</td>
</tr>
<tr>
<td>Null</td>
<td>10, 11, 14, 18</td>
</tr>
<tr>
<td>Orify</td>
<td>4, 15</td>
</tr>
<tr>
<td>Partial</td>
<td>8, 10, 13, 15, 16</td>
</tr>
<tr>
<td>partial</td>
<td>15, 16</td>
</tr>
<tr>
<td>Reduce</td>
<td>16</td>
</tr>
<tr>
<td>Reduce_Right</td>
<td>8, 10, 13, 16, 16</td>
</tr>
</tbody>
</table>