Package ‘lenses’

March 6, 2019

Version 0.0.3

Title Elegant Data Manipulation with Lenses


License MIT + file LICENSE

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 6.0.1

URL http://cfhammill.github.io/lenses,
     https://github.com/cfhammill/lenses

BugReports https://github.com/cfhammill/lenses/issues

Suggests testthat

Imports magrittr, tidyselect, rlang

Collate 'verbs.R' 'lens.R' 'array-lenses.R' 'base-lenses.R' 'utils.R'
     'dataframe-lenses.R' 'utils.pipe.R'

NeedsCompilation no

Author Chris Hammill [aut, cre, trl],
     Ben Darwin [aut, trl]

Maintainer Chris Hammill <cfhammill@gmail.com>

Repository CRAN

Date/Publication 2019-03-06 14:40:03 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes_l</td>
<td>3</td>
</tr>
<tr>
<td>attr_l</td>
<td>3</td>
</tr>
<tr>
<td>body_l</td>
<td>4</td>
</tr>
<tr>
<td>class_l</td>
<td>4</td>
</tr>
<tr>
<td>colnames_l</td>
<td>5</td>
</tr>
<tr>
<td>cols_l</td>
<td>5</td>
</tr>
<tr>
<td>cond_il</td>
<td>6</td>
</tr>
<tr>
<td>c_l</td>
<td>6</td>
</tr>
<tr>
<td>diag_l</td>
<td>7</td>
</tr>
<tr>
<td>dimnames_l</td>
<td>7</td>
</tr>
<tr>
<td>dim_l</td>
<td>8</td>
</tr>
<tr>
<td>drop_while_il</td>
<td>8</td>
</tr>
<tr>
<td>env_l</td>
<td>9</td>
</tr>
<tr>
<td>filter_il</td>
<td>9</td>
</tr>
<tr>
<td>filter_l</td>
<td>10</td>
</tr>
<tr>
<td>first_l</td>
<td>10</td>
</tr>
<tr>
<td>formals_l</td>
<td>11</td>
</tr>
<tr>
<td>id_l</td>
<td>11</td>
</tr>
<tr>
<td>indexes_l</td>
<td>12</td>
</tr>
<tr>
<td>index_l</td>
<td>12</td>
</tr>
<tr>
<td>last_l</td>
<td>13</td>
</tr>
<tr>
<td>lens</td>
<td>13</td>
</tr>
<tr>
<td>levels_l</td>
<td>14</td>
</tr>
<tr>
<td>lower_tri_l</td>
<td>15</td>
</tr>
<tr>
<td>map_l</td>
<td>15</td>
</tr>
<tr>
<td>names_l</td>
<td>16</td>
</tr>
<tr>
<td>oscpe</td>
<td>16</td>
</tr>
<tr>
<td>over</td>
<td>17</td>
</tr>
<tr>
<td>over_map</td>
<td>17</td>
</tr>
<tr>
<td>over_with</td>
<td>18</td>
</tr>
<tr>
<td>reshape_l</td>
<td>18</td>
</tr>
<tr>
<td>rev_l</td>
<td>19</td>
</tr>
<tr>
<td>rownames_l</td>
<td>19</td>
</tr>
<tr>
<td>rows_l</td>
<td>20</td>
</tr>
<tr>
<td>select_l</td>
<td>20</td>
</tr>
<tr>
<td>send</td>
<td>21</td>
</tr>
<tr>
<td>send_over</td>
<td>21</td>
</tr>
<tr>
<td>set</td>
<td>22</td>
</tr>
<tr>
<td>slab_l</td>
<td>22</td>
</tr>
<tr>
<td>slice_l</td>
<td>23</td>
</tr>
<tr>
<td>slot_l</td>
<td>23</td>
</tr>
<tr>
<td>take_l</td>
<td>24</td>
</tr>
<tr>
<td>take_while_il</td>
<td>24</td>
</tr>
<tr>
<td>to_l</td>
<td>25</td>
</tr>
<tr>
<td>transpose_l</td>
<td>25</td>
</tr>
<tr>
<td>t_l</td>
<td>26</td>
</tr>
</tbody>
</table>
**attributes_l**

The lens equivalent of attributes and attributes<-

**Usage**

attributes_l

**Format**

An object of class lens of length 2.

**Examples**

```r
(x <- structure(1:10, important = "attribute"))
view(x, attributes_l)
set(x, attributes_l, list(important = "feature"))
```

**attr_l**

Construct a lens into an attribute

**Description**

The lens version of attr and attr<-

**Usage**

attr_l(attrib)

**Arguments**

* attrib A length one character vector indicating the attribute to lens into.

**Examples**

```r
(x <- structure(1:10, important = "attribute"))
view(x, attr_l("important"))
set(x, attr_l("important"), "feature")
```
**body_l**  
*Body lens*

**Description**
A lens into the body of a function. The lens equivalent of `body` and `body<-`. You probably shouldn’t use this.

**Usage**
```r
body_l
```

**Format**
An object of class `lens` of length 2.

**Examples**
```r
incR <- function(x) x + R
view(incR, body_l)
incT <- set(incR, body_l, quote(x + T))
incT(1)
```

**class_l**  
*Class lens*

**Description**
A lens into the class of an object. Lens equivalent of `class` and `class<-`.

**Usage**
```r
class_l
```

**Format**
An object of class `lens` of length 2.

**Examples**
```r
x <- 1:10
view(x, class_l)
set(x, class_l, "super_integer")
```
colnames_l

A lens into the column names of an object

Description

The lens version of colnames and colnames<-

Usage

colnames_l

Format

An object of class lens of length 2.

Examples

x <- matrix(1:4, ncol = 2)
colnames(x) <- c("first", "second")
x

view(x, colnames_l)
set(x, colnames_l, c("premiere", "deuxieme"))

cols_l

Column lens

Description

Create a lens into a set of columns

Usage

cols_l(cols, drop = FALSE)

Arguments

cols the columns to focus on
drop whether or not to drop dimensions with length 1

Examples

x <- matrix(1:4, ncol = 2)
colnames(x) <- c("first", "second")
x

view(x, cols_l(1))
view(x, cols_l("second"))
set(x, cols_l(1), c(20, 40))
cond_il  

**Conditional lens**

**Description**

`view` is equivalent to `Filter(f, d)`, `set` replaces elements that satisfy `f` with elements of `x`.

**Usage**

```r
cond_il(f)
```

**Arguments**

- `f` the predicate (logical) function

**Details**

This lens is illegal because `set-view` is not satisfied, multiple runs of the same lens will reference potentially different elements.

c_l  

**Convenient lens composition**

**Description**

A lens version of `purrr::pluck`. Takes a series element indicators and creates a composite lens.

**Usage**

```r
c_l(...)```

**Arguments**

- `...` index vectors or lenses

**Details**

- length one vectors are converted to `index_l`,
- length one logical vectors and numeric vectors that are negative are converted to `indexes_l`,
- larger vectors are converted to `indexes_l`,
- lenses are composed as is.

See examples for more

**Examples**

```r
view(iris, c_l("Petal.Length", 10:20, 3))
sepal_l <- index("Sepal.Length")
view(iris, c_l(sepal_l, id_l, 3))```
**diag_l**

_Lens into the diagonal of a matrix_

**Description**

A lens into a matrix’s diagonal elements

**Usage**

```r
diag_l
```

**Format**

An object of class `lens` of length 2.

---

**dimnames_l**

_Dimnames lens_

**Description**

A lens into the dimnames of an object. Lens equivalent of `dimnames` and `dimnames<-`.

**Usage**

```r
dimnames_l
```

**Format**

An object of class `lens` of length 2.

**Examples**

```r
x <- matrix(1:4, ncol = 2)
colnames(x) <- c("first", "second")
x

view(x, dimnames_l)
set(x, dimnames_l, list(NULL, c("premier", "deuxième")))
```
dim_l

Description
A lens into an object's dimensions

Usage
dim_l

Format
An object of class `lens` of length 2.

Examples
```r
x <- 1:10
(y <- set(x, dim_l, c(2, 5)))
view(y, dim_l)
```

drop_while_il

Description
A lens into all elements starting from the first element that doesn’t satisfy a predicate. Essentially the complement of `take_while_il`

Usage
drop_while_il(f)

Arguments
```r
f the predicate (logical) function
```
**env_l**

*Environment lens*

**Description**

A lens into the environment of an object. This is the lens version of `environment` and `environment<-`.

**Usage**

`env_l`

**Format**

An object of class `lens` of length 2.

**Examples**

```r
x <- 10
f <- (function(){x <- 2; function() x + 1})()
f

f()
view(f, env_l)$x

g <- over(f, env_l, parent.env)
g()
```

---

**filter_il**

*Filter lens*

**Description**

Create an illegal lens into the result of a filter. Arguments are interpreted with non-standard evaluation as in `dplyr::filter`.

**Usage**

`filter_il(...)`

**Arguments**

`...` unquoted NSE filter arguments

**Examples**

```r
head(view(iris, filter_il(Species == "setosa")))
head(over(iris,
         filter_il(Species == "setosa") %>% select(-Species),
         function(x) x + 10))
```
filter_l  

Filter lens

Description
Create a lawful lens into the result of a filter. This focuses only columns not involved in the filter condition.

Usage
filter_l(...)

Arguments

... unquoted NSE filter arguments

Examples
head(view(iris, filter_l(Species == "setosa"))) # Note Species is not seen
head(over(iris, filter_l(Species == "setosa"), function(x) x + 10))

first_l  

A lens into the first element

Description
Lens version of x[1] and x[[1]] <- val x <- 1:10 view(x, first_l) set(x, first_l, 50)
[[1]]: R:1 [[1]: R:1

Usage

first_l

Format
An object of class lens of length 2.
**formals_l**

Formals lens

Description
A lens equivalent of `formals` and `formals<->`, allowing you to change the formal arguments of a function. As with `body_l` you probably shouldn’t use this.

Usage
`formals_l`

Format
An object of class `lens` of length 2.

Examples
```r
f <- function(x) x + y + 7
view(f, formals_l)

g <- set(f, formals_l, list(x = 1, y = 2))
g()
```

---

**id_l**

The identity (trivial lens)

Description
This lens focuses on the whole object

Usage
`id_l`

Format
An object of class `lens` of length 2.

Examples
```r
x <- 1:10
view(x, id_l)
head(set(x, id_l, iris))
```
indexes_l

Construct a lens into a subset of an object

Description
This is the lens version of [

Usage
indexes_l(els)
indexes(els)

Arguments
els a subset vector, can be integer, character of logical, pointing to one or more elements of the object

Functions
• indexes: shorthand

Examples
x <- 1:10
view(x, indexes_l(3:5))
set(x, indexes_l(c(1,10)), NA)
head(view(iris, indexes_l(c("Sepal.Length", "Species")))))

index_l

Construct a lens into an index/name

Description
This is the lens version of []

Usage
index_l(els)
index(els)

Arguments
el The element the lens should point to can be an integer or name.
Functions

- index: shorthand

Examples

```r
x <- 1:10
view(x, index_l(1))
set(x, index(5), 50)
head(view(iris, index(2)))
```

---

**last_l**

*A lens into the last element*

Description

Lens version of `x[[length(x)]]` and `x[[length(x)]] <- val`

`[[length(x)]]: R:length(x) [[length(x)]]: R:length(x)`

Usage

```
last_l
```

Format

An object of class `lens` of length 2.

Examples

```r
x <- 1:10
view(x, last_l)
set(x, last_l, 50)
```

---

**lens**

*Construct a lens*

Description

A lens represents the process of focusing on a specific part of a data structure. We represent this via a view function and an set function, roughly corresponding to object-oriented "getters" and "setters" respectively. Lenses can be composed to access or modify deeply nested structures.

Usage

```
lens(view, set, getter = FALSE)
```
Arguments

view  A function that takes a data structure of a certain type and returns a subpart of that structure

set  A function that takes a data structure of a certain type and a value and returns a new data structure with the given subpart replaced with the given value. Note that set should not modify the original data.

getter  Default is FALSE, if TRUE the created lens cannot be set into.

Details

Lenses are popular in functional programming because they allow you to build pure, compositional, and re-usable "getters" and "setters".

As noted in the README, using lens directly incurs the following obligations (the "Lens laws"):

1. Get-Put: If you get (view) some data with a lens, and then modify (set) the data with that value, you get the input data back.
2. Put-Get: If you put (set) a value into some data with a lens, then get that value with the lens, you get back what you put in.
3. Put-Put: If you put a value into some data with a lens, and then put another value with the same lens, it's the same as only doing the second put.

"Lenses" which do not satisfy these properties should be documented accordingly. By convention, such objects present in this library are suffixed by "_il" ("illegal lens").

Examples

third_l <- lens(view = function(d) d[[3]],
                set = function(d, x)( d[[3]] <- x; d ))
view(1:10, third_l) # returns 3
set(1:10, third_l, 10) # returns c(1:2, 10, 4:10)

levels_l  

Description

A lens into the levels of an object. Usually this is factor levels. Lens equivalent of levels and levels<-.

Usage

levels_l

Format

An object of class lens of length 2.
lower_tri_l

**Examples**

```r
x <- factor(c("A", "B"))
view(x, levels_l)
set(x, levels_l, c("A", "B"))
```

---

**lower_tri_l**  
**Lens into lower diagonal elements**

**Description**

Create a lens into the lower diagonal elements of a matrix

**Usage**

```r
lower_tri_l(diag = FALSE)
```

**Arguments**

- `diag` whether or not to include the diagonal

**Examples**

```r
(x <- matrix(1:9, ncol = 3))
view(x, lower_tri_l())
view(x, lower_tri_l(diag = TRUE))
set(x, lower_tri_l(), c(100, 200, 300))
```

---

**map_l**  
**Promote a lens to apply to each element of a list**

**Description**

Create a new lens that views and sets each element of the list.

**Usage**

```r
map_l(l)
```

**Arguments**

- `l` the lens to promote

**Details**

Uses `lapply` under the hood for `view` and `mapply` under the hood for `set`. This means that `set` can be given a list of values to set, one for each element. If the input or update are lists this lens always returns a list. If the input and update are vectors this lens will return a vector.
Examples
(ex <- replicate(10, sample(1:5), simplify = FALSE))
view(ex, map_l(index(1)))
set(ex, map_l(index(1)), 11:20)

names_l  
A lens into the names of an object

Description
The lens versions of names and names<-

Usage
names_l

Format
An object of class lens of length 2.

Examples
view(iris, names_l)
head(set(iris, names_l, LETTERS[1:5]))

oscope  
Bind data to a lens

Description
To flatten lens composition, you can prespecify the data the lens with be applied to by constructing an objectoscope. These can be integrated easily with normal data pipelines.

Usage
oscope(d, l = id_l)

Arguments
  d  The data for interest
  l  The lens to bind the data to. Defaults to the identity lens

Examples
list(a = 5, b = 1:3, c = 8) %>%
  oscpe()  %.
  index_l("b")  %.
  index_l(1)  %>
  set(10)
\textbf{over} \hspace{1cm} \textit{Map a function over a lens}

**Description**
Get the data pointed to by a lens, apply a function and replace it with the result.

**Usage**
\texttt{over(d, l, f)}

**Arguments**
- \texttt{d} \hspace{1cm} the data (or an \texttt{oscope})
- \texttt{l} \hspace{1cm} the lens (or the function if \texttt{d} is an \texttt{oscope})
- \texttt{f} \hspace{1cm} the function (or nothing if \texttt{d} is an \texttt{oscope})

**Examples**
\begin{verbatim}
third_l <- index(3)
over(1:5, third_l, function(x) x + 2)
# returns c(1:2, 5, 4:5)
\end{verbatim}

---

\textbf{over_map} \hspace{1cm} \textit{Map a function over a list lens}

**Description**
Apply the specified function to each element of the subobject.

**Usage**
\texttt{over_map(d, l, f)}

**Arguments**
- \texttt{d} \hspace{1cm} the data
- \texttt{l} \hspace{1cm} the lens
- \texttt{f} \hspace{1cm} the function to use, potentially a \texttt{~} specified anonymous function.
over_with  

Map a function over an in scope lens

Description

Apply the specified function with named elements of the viewed data in scope. Similar to `dplyr::mutate`

Usage

```r
over_with(d, l, f)
```

Arguments

- `d`: the data
- `l`: the lens
- `f`: the function to use, potentially a `~` specified anonymous function. The function body is quoted, and evaluated with `rlang::eval_tidy(...)`, `data = view(d, l))`

Examples

```r
iris %>% over_with(id, ~ Sepal.Length)
```

reshape_l  

Lens into a new dimension(s)

Description

Construct a lens that is a view of the data with a new set of dimensions. Both `view` and `set` check that the new dimensions match the number of elements of the data.

Usage

```r
reshape_l(dims)
```

Arguments

- `dims`: a vector with the new dimensions

Examples

```r
x <- 1:9
view(x, reshape_l(c(3,3)))
set(x, reshape_l(c(3,3)) %>% diag_l(100))
```
rev_l

Reverse lens

Description

Lens into the reverse of an object.

Usage

rev_l

Format

An object of class lens of length 2.

Examples

```r
x <- 1:10
view(x, rev_l)
set(x, rev_l, 11:20)
```

rownames_l

A lens into the row names of an object

Description

The lens version of rownames and rownames<-

Usage

rownames_l

Format

An object of class lens of length 2.

Examples

```r
x <- matrix(1:4, ncol = 2)
rownames(x) <- c("first", "second")
x
view(x, rownames_l)
set(x, rownames_l, c("premiere", "deuxieme"))
```
‘rows_l’ Row lens

**Description**
Create a lens into a set of rows

**Usage**

```
rows_l(rows, drop = FALSE)
```

**Arguments**
- **rows**: the rows to focus on
- **drop**: whether or not to drop dimensions with length 1

**Examples**

```r
x <- matrix(1:4, ncol = 2)  
rownames(x) <- c("first", "second")  
x

view(x, rows_l(1))  
view(x, rows_l("second"))  
set(x, rows_l(1), c(20, 40))
```

---

‘select_l’ Tidyselect elements by name

**Description**
Create a lens into a named collection. On set names of the input are not changed. This generalizes `dplyr::select` to arbitrary named collections and allows updating.

**Usage**

```
select_l(...)  
```

**Arguments**
- **...**: An expression to be interpreted by `tidyselect::vars_select` which is the same interpreter as `dplyr::select`

**Examples**

```r
lets <- setNames(seq_along(LETTERS), LETTERS)  
set(lets, select_l(G:F, A, B), 1:4) # A and B are 3,4 for a quick check
```
send

\textit{Set one lens to the view of another}

\textbf{Description}

Set one lens to the view of another

\textbf{Usage}

send(d, l, m)

\textbf{Arguments}

\begin{itemize}
  \item \texttt{d} \quad \text{the data}
  \item \texttt{l} \quad \text{the lens to view through}
  \item \texttt{m} \quad \text{the lens to set into}
\end{itemize}

send\_over

\textit{Set one lens to the view of another (transformed)}

\textbf{Description}

Set one lens to the view of another (transformed)

\textbf{Usage}

send\_over(d, l, m, f)

\textbf{Arguments}

\begin{itemize}
  \item \texttt{d} \quad \text{the data}
  \item \texttt{l} \quad \text{the lens to view through}
  \item \texttt{m} \quad \text{the lens to set into}
  \item \texttt{f} \quad \text{the function to apply to the viewed data}
\end{itemize}
set  
Modify data with a lens

Description

Set the subcomponent of the data referred to by a lens with a new value. See lens for details. Merely dispatches to the set component of the lens.

Usage

`set(d, l, x)`

Arguments

- `d`: the data, or an `oscope`
- `l`: the lens, or in the case of an `oscope`, the replacement
- `x`: the replacement value, or nothing in the case of an `oscope`

slab_l  
Slab lens

Description

Create a lens into a chunk of an array (hyperslab). Uses the same syntactic rules as `[.].

Usage

`slab_l(..., drop = FALSE)`

Arguments

- `...`: arguments as they would be passed to `[ for example `x[3, 5, 7]`
- `drop`: whether or not to drop dimensions with length 1. Only applies to `view`.

Examples

```r
(x <- matrix(1:4, ncol = 2))
view(x, slab_l(2,)) # x[2,, drop = FALSE]
view(x, slab_l(2, 2)) # x[2,2, drop = FALSE]
set(x, slab_l(1,1:2), c(10, 20))
```
slice_l

Slice lens

Description
Create a lens into a specific slice of a specific dimension of a multidimensional object. Not to be confused with dplyr slice.

Usage
slice_l(dimension, slice, drop = FALSE)

Arguments
- dimension: the dimension to slice
- slice: the slice index
- drop: whether or not to drop dimensions with length 1. Only applies to view.

Examples
(x <- matrix(1:4, ncol = 2))
view(x, slice_l(1, 2)) # x[2,. , drop = FALSE]
view(x, slice_l(2, 2)) # x[.2, , drop = FALSE]
set(x, slice_l(1,1), c(10,20))

slot_l
Slot lens

Description
The lens equivalent of @ and @<- for getting and setting S4 object slots.

Usage
slot_l(slot)

Arguments
- slot: the name of the slot

Examples
new_class <- setClass("new_class", slots = c(x = "numeric"))
(x <- new_class())
view(x, slot_l("x"))
set(x, slot_l("x"), 1:10)
take_l

Construct a lens into a prefix of a vector

Description
This constructs a lens into the first n elements of an object or the if negative indexing is used, as many as length(x) - n.

Usage
take_l(n)

Arguments
n number of elements to take, or if negative the number of elements at the end to not take.

Examples
x <- 1:10
view(x, take_l(3))
view(x, take_l(-7))
set(x, take_l(2), c(100,200))
set(x, take_l(-8), c(100,200))

take_while_il
Conditional head lens

Description
A lens into the elements from the beginning of a structure until the last element that satisfies a predicate.

Usage
take_while_il(f)

Arguments
f the predicate (logical) function

Details
This lens is illegal because set-view is not satisfied, multiple runs of the same lens will reference potentially different elements.
to_l

Promote a function to a getter lens

Description

Create a getter lens from a function.

Usage

to_l(f)

Arguments

f  The function to promote.

Examples

# This wouldn't make sense as a general legal lens, but fine as a 'getter'
sqrt_l <- to_l(sqrt)
iris_root <- index(1) .% index(1) .% sqrt_l

sqrt(iris[[1]][[1]])
iris %>% view(iris_root)
tryCatch(iris %>% set(iris_root, 2)
  , error = function(e) "See, can't do that")

transpose_l

Lens into a list of rows

Description

A lens that creates a list-of-rows view of a data.frame

Usage

transpose_l

Format

An object of class lens of length 2.
t_l  Matrix transpose lens

Description

Lens into the transpose of a matrix

Usage

t_l

Format

An object of class lens of length 2.

Examples

```r
(x <- matrix(1:14, ncol = 2))
view(x, t_l)
set(x, t_l, matrix(11:14, ncol = 2))
```

unlist_l  Unlist lens

Description

A lens between a list and an unrecursively unlisted object.

Usage

unlist_l

Format

An object of class lens of length 2.

Examples

```r
(x <- list(x = list(y = 1:10)))
view(x, unlist_l)
set(x, unlist_l ++) unlist_l, rep("hello", 10))
```
upper_tri_l

Lens into upper diagonal elements

Description

Create a lens into the upper diagonal elements of a matrix

Usage

```r
upper_tri_l(diag = FALSE)
```

Arguments

diag whether or not to include the diagonal

```r
x <- matrix(1:9, ncol = 3)
view(x, upper_tri_l())
view(x, upper_tri_l(diag = TRUE))
set(x, upper_tri_l(), c(100, 200, 300))
```

view

View data with a lens

Description

Get the subcomponent of the data referred to by a lens. This function merely dispatches to the view component of the lens.

Usage

```r
view(d, l)
```

Arguments

d the data
l the lens
Compose lenses

Description

Compose two lenses to produce a new lens which represents focussing first with the first lens, then with the second. A view using the resulting composite lens will first view using the first, then the second, while an set will view via the first lens, set into the resulting piece with the second, and then replace the updated structure in the first with set. Lens composition is analogous to the syntax of object-oriented programming or to a flipped version of function composition.

Usage

\[ l \circ m \]

Arguments

- \( l \) the first lens (or an oscope)
- \( m \) the second lens

Examples

```r
lst <- list(b = c(3,4,5))
lns <- index_l("b") \circ index_l(2)
lst %>>% view(lns) \# returns 4
lst %>>% set(lns, 1) \# returns list(b = c(3,2,5))
lst \# returns list(b = c(3,4,5))
```
Index

*Topic datasets
  attributes_l, 3
  body_l, 4
  class_l, 4
  colnames_l, 5
  diag_l, 7
  dim_l, 8
  dimnames_l, 7
  env_l, 9
  first_l, 10
  formals_l, 11
  id_l, 11
  last_l, 13
  levels_l, 14
  names_l, 16
  rev_l, 19
  rownames_l, 19
  t_l, 26
  transpose_l, 25
  unlist_l, 26
  %., 28
  attr_l, 3
  attributes, 3
  attributes<-, 3
  attributes_l, 3
  body, 4
  body<-, 4
  body_l, 4, 11
  c_l, 6
  class, 4
  class<-, 4
  class_l, 4
  colnames_l, 5
  cols_l, 5
  cond_Il, 6
  diag_l, 7
  dim_l, 8
  dimnames, 7
  dimnames<-, 7
  dimnames_l, 7
dplyr::filter, 9
dplyr::mutate, 18
dplyr::select, 20
  drop_while_il, 8
  env_l, 9
  environment, 9
  environment<-, 9
  filter_il, 9
  filter_l, 10
  first_l, 10
  formals, 11
  formals<-, 11
  formals_l, 11
  id_l, 11
  index (index_l), 12
  index_l, 6, 12
  indexes (indexes_l), 12
  indexes_l, 6, 12
  lapply, 15
  last_l, 13
  lens, 13, 22, 28
  levels, 14
  levels<-, 14
  levels_l, 14
  lower_tri_l, 15
  map_l, 15
  mapply, 15
  names_l, 16
  oscope, 16, 17, 22, 28
  over, 17
over_map, 17
over_with, 18

purrr::pluck, 6

reshape_l, 18
rev, 19
rev_l, 19
rownames_l, 19
rows_l, 20

select_l, 20
send, 21
send_over, 21
set, 6, 15, 18, 20, 22
slab_l, 22
slice_l, 23
slot_l, 23

t_l, 26
take_l, 24
take_while_il, 8, 24
tidyselect::vars_select, 20
to_l, 25
transpose_l, 25

unlist, 26
unlist_l, 26
upper_tri_l, 27

view, 6, 15, 18, 23, 27