

Apply functions with purrr : : CHEAT SHEET



Map Functions

ONE LIST

map(.x, .f, ...) Apply a function to each element of a list or vector, return a list.

```
x <- list(1:10, 11:20, 21:30)
l1 <- list(x = c("a", "b"), y = c("c", "d"))
map(l1, sort, decreasing = TRUE)
```



TWO LISTS

map2(.x, .y, .f, ...) Apply a function to pairs of elements from two lists or vectors, return a list.

```
y <- list(1, 2, 3); z <- list(4, 5, 6); l2 <- list(x = "a", y = "z")
map2(x, y, ~ .x * .y)
```



MANY LISTS

pmap(.l, .f, ...) Apply a function to groups of elements from a list of lists or vectors, return a list.

```
pmap(list(x, y, z), ~ ..1 * (..2 + ..3))
```



LISTS AND INDEXES

imap(.x, .f, ...) Apply .f to each element and its index, return a list.

```
imap(y, ~ paste0(.y, ":", .x))
```



map_dbl(.x, .f, ...) Return a double vector.
map_dbl(x, mean)

map_int(.x, .f, ...) Return an integer vector.
map_int(x, length)

map_chr(.x, .f, ...) Return a character vector.
map_chr(l1, paste, collapse = "")

map_lgl(.x, .f, ...) Return a logical vector.
map_lgl(x, is.integer)

map_dfc(.x, .f, ...) Return a data frame created by column-binding.
map_dfc(l1, rep, 3)

map_dfr(.x, .f, ..., .id = NULL) Return a data frame created by row-binding.
map_dfr(x, summary)

walk(.x, .f, ...) Trigger side effects, return invisibly.
walk(x, print)

map2_dbl(.x, .y, .f, ...) Return a double vector.
map2_dbl(y, z, ~ .x / .y)

map2_int(.x, .y, .f, ...) Return an integer vector.
map2_int(y, z, `+`)

map2_chr(.x, .y, .f, ...) Return a character vector.
map2_chr(l1, l2, paste, collapse = ";", sep = ":")

map2_lgl(.x, .y, .f, ...) Return a logical vector.
map2_lgl(l2, l1, `~ %in%`)

map2_dfc(.x, .y, .f, ...) Return a data frame created by column-binding.
map2_dfc(l1, l2, ~ as.data.frame(c(x, y)))

map2_dfr(.x, .y, .f, ..., .id = NULL) Return a data frame created by row-binding.
map2_dfr(l1, l2, ~ as.data.frame(c(x, y)))

walk2(.x, .y, .f, ...) Trigger side effects, return invisibly.
walk2(objs, paths, save)

pmap_dbl(.l, .f, ...) Return a double vector.
pmap_dbl(list(y, z), ~ .x / .y)

pmap_int(.l, .f, ...) Return an integer vector.
pmap_int(list(y, z), `+`)

pmap_chr(.l, .f, ...) Return a character vector.
pmap_chr(list(l1, l2), paste, collapse = ";", sep = ":")

pmap_lgl(.l, .f, ...) Return a logical vector.
pmap_lgl(list(l2, l1), `~ %in%`)

pmap_dfc(.l, .f, ...) Return a data frame created by column-binding.
pmap_dfc(list(l1, l2), ~ as.data.frame(c(x, y)))

pmap_dfr(.l, .f, ..., .id = NULL) Return a data frame created by row-binding.
pmap_dfr(list(l1, l2), ~ as.data.frame(c(x, y)))

pwalk(.l, .f, ...) Trigger side effects, return invisibly.
pwalk(list(objs, paths), save)

imap_dbl(.x, .f, ...) Return a double vector.
imap_dbl(y, ~ .y)

imap_int(.x, .f, ...) Return an integer vector.
imap_int(y, ~ .y)

imap_chr(.x, .f, ...) Return a character vector.
imap_chr(y, ~ paste0(.y, ":", .x))

imap_lgl(.x, .f, ...) Return a logical vector.
imap_lgl(l1, ~ is.character(.y))

imap_dfc(.x, .f, ...) Return a data frame created by column-binding.
imap_dfc(l2, ~ as.data.frame(c(x, y)))

imap_dfr(.x, .f, ..., .id = NULL) Return a data frame created by row-binding.
imap_dfr(l2, ~ as.data.frame(c(x, y)))

iwalk(.x, .f, ...) Trigger side effects, return invisibly.
iwalk(z, ~ print(paste0(.y, ":", .x)))

Function Shortcuts

Use `~ .` with functions like **map()** that have single arguments.

```
map(l, ~ . + 2)
becomes
map(l, function(x) x + 2)
```

Use `~ .x .y` with functions like **map2()** that have two arguments.

```
map2(l, p, ~ .x + .y)
becomes
map2(l, p, function(l, p) l + p)
```

Use `~ ..1 ..2 ..3` etc with functions like **pmap()** that have many arguments.

```
pmap(list(a, b, c), ~ ..3 + ..1 - ..2)
becomes
pmap(list(a, b, c), function(a, b, c) c + a - b)
```

Use `~ .x .y` with functions like **imap()**. `.x` will get the list value and `.y` will get the index, or name if available.

```
imap(list(a, b, c), ~ paste0(.y, ":", .x))
outputs "index: value" for each item
```

Use a **string** or an **integer** with any map function to index list elements by name or position. **map(l, "name")** becomes **map(l, function(x) x[["name"]])**





Work with Lists

Filter

keep(.x, .p, ...)
Select elements that pass a logical test.
Conversely, **discard()**.
`keep(x, is.na)`

compact(.x, .p = identity)
Drop empty elements.
`compact(x)`

head_while(.x, .p, ...)
Return head elements until one does not pass.
Also **tail_while()**.
`head_while(x, is.character)`

detect(.x, .f, ..., dir = c("forward", "backward"), .right = NULL, .default = NULL)
Find first element to pass.
`detect(x, is.character)`

detect_index(.x, .f, ..., dir = c("forward", "backward"), .right = NULL)
Find index of first element to pass.
`detect_index(x, is.character)`

every(.x, .p, ...)
Do all elements pass a test?
`every(x, is.character)`

some(.x, .p, ...)
Do some elements pass a test?
`some(x, is.character)`

none(.x, .p, ...)
Do no elements pass a test?
`none(x, is.character)`

has_element(.x, .y)
Does a list contain an element?
`has_element(x, "foo")`

vec_depth(x)
Return depth (number of levels of indexes).
`vec_depth(x)`

Index

pluck(.x, ..., .default=NULL)
Select an element by name or index. Also **attr_getter()** and **chuck()**.
`pluck(x, "b")`
`x %>% pluck("b")`

assign_in(x, where, value)
Assign a value to a location using pluck selection.
`assign_in(x, "b", 5)`
`x %>% assign_in("b", 5)`

modify_in(.x, .where, .f)
Apply a function to a value at a selected location.
`modify_in(x, "b", abs)`
`x %>% modify_in("b", abs)`

Reshape

flatten(.x)
Remove a level of indexes from a list.
Also **flatten_chr()** etc.
`flatten(x)`

array_tree(array, margin = NULL)
Turn array into list.
Also **array_branch()**.
`array_tree(x, margin = 3)`

cross2(.x, .y, .filter = NULL)
All combinations of .x and .y. Also **cross()**, **cross3()**, and **cross_df()**.
`cross2(1:3, 4:6)`

transpose(.l, .names = NULL)
Transposes the index order in a multi-level list.
`transpose(x)`

set_names(x, nm = x)
Set the names of a vector/list directly or with a function.
`set_names(x, c("p", "q", "r"))`
`set_names(x, tolower)`

Modify

modify(.x, .f, ...)
Apply a function to each element. Also **modify2()**, and **imodify()**.
`modify(x, ~.+ 2)`

modify_at(.x, .at, .f, ...)
Apply a function to selected elements. Also **map_at()**.
`modify_at(x, "b", ~.+ 2)`

modify_if(.x, .p, .f, ...)
Apply a function to elements that pass a test. Also **map_if()**.
`modify_if(x, is.numeric, ~.+ 2)`

modify_depth(.x, .depth, .f, ...)
Apply function to each element at a given level of a list. Also **map_depth()**.
`modify_depth(x, 2, ~.+ 2)`

Combine

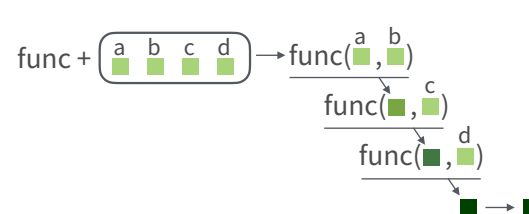
append(x, values, after = length(x))
Add values to end of list.
`append(x, list(d = 1))`

prepend(x, values, before = 1)
Add values to start of list.
`prepend(x, list(d = 1))`

splice(...)
Combine objects into a list, storing S3 objects as sub-lists.
`splice(x, y, "foo")`

Reduce

reduce(.x, .f, ..., .init, .dir = c("forward", "backward"))
Apply function recursively to each element of a list or vector. Also **reduce2()**.
`reduce(x, sum)`



List-Columns

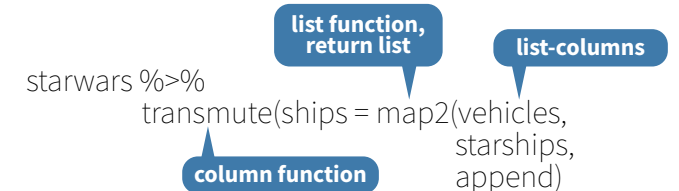
max	seq
3	<int [3]>
4	<int [4]>
5	<int [5]>

List-columns are columns of a data frame where each element is a list or vector instead of an atomic value. Columns can also be lists of data frames. See **tidyr** for more about nested data and list columns.

WORK WITH LIST-COLUMNS

Manipulate list-columns like any other kind of column, using **dplyr** functions like **mutate()** and **transmute()**. Because each element is a list, use **map functions** within a column function to manipulate each element.

map(), map2(), or pmap() return lists and will create new list-columns.



Suffixed map functions like **map_int()** return an atomic data type and will **simplify list-columns into regular columns**.

