

Aplique funciones con purrr : : GUÍA RÁPIDA

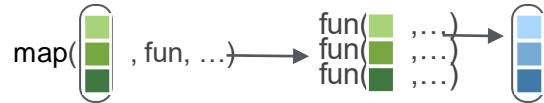


Funciones de mapa

UNA LISTA

map(x, .f, ...) Aplique una función a cada elemento de una lista o vector y devuelva una lista.

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



map_dbl(x, .f, ...)
Devuelve un vector doble.
map_dbl(x, mean)

map_int(x, .f, ...)
Devolver un vector entero.
map_int(x, length)

map_chr(x, .f, ...)
Devolver un vector de caracteres.
map_chr(l1, paste, collapse = "")

map_lgl(x, .f, ...)
Devolver un vector lógico.
map_lgl(x, is.integer)

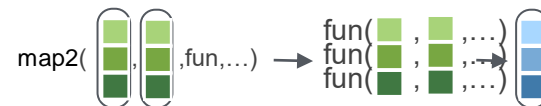
map_vec(x, .f, ...)
Devuelve un vector que es del tipo común más simple.
map_vec(l1, paste, collapse = "")

walk(x, .f, ...) Desencadenan efectos secundarios, regresan de forma invisible.
walk(x, print)

DOS LISTAS

map2(x, y, .f, ...) Aplique una función a pares de elementos de dos listas o vectores, devuelva una lista.

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



map2_dbl(x, y, .f, ...)
Devuelve un vector doble.
map2_dbl(y, z, ~ .x / .y)

map2_int(x, y, .f, ...)
Devuelve un vector entero.
map2_int(y, z, +)

map2_chr(x, y, .f, ...)
Devuelve un vector de caracteres.
map2_chr(l1, l2, paste, collapse = "", sep = ":")

map2_lgl(x, y, .f, ...) Devuelve un vector lógico.
map2_lgl(l2, l1, `in`)

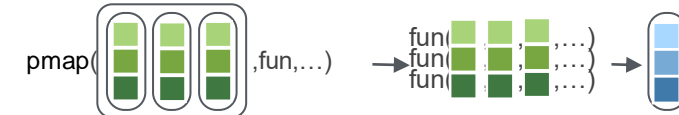
map2_vec(x, y, .f, ...)
Devuelve un vector que es del tipo común más simple.
map2_vec(l1, l2, paste, collapse = "", sep = ":")

walk2(x, y, .f, ...) Desencadena efectos secundarios, regresa de forma invisible.
walk2(objs, paths, save)

MUCHAS LISTAS

pmap(.l, .f, ...) Aplique una función a grupos de elementos de una lista de listas o vectores, devuelva una lista.

```
pmap(
  list(x, y, z),
  function(first, second, third) first * (second + third)
)
```



pmap_dbl(.l, .f, ...)
Devuelve un vector doble.
pmap_dbl(list(y, z), ~ .x / .y)

pmap_int(.l, .f, ...)
Devuelve un vector entero.
pmap_int(list(y, z), `+`)

pmap_chr(.l, .f, ...)
Devuelve un vector de caracteres.
pmap_chr(list(l1, l2), paste, collapse = "", sep = ":")

pmap_lgl(.l, .f, ...)
Devuelve un vector lógico.
pmap_lgl(list(l2, l1), `in`)

pmap_vec(.l, .f, ...)
Devuelve un vector que es del tipo común más simple.
pmap_vec(list(l1, l2), paste, collapse = "", sep = ":")

pwalk(.l, .f, ...) Desencadena efectos secundarios, regresa de forma invisible.
pwalk(list(objs, paths), save)

Atajos de función

Use `\(x)` con funciones como `map()` que tienen argumentos únicos.

```
map(l, \(x) x + 2)
# se convierte
map(l, function(x) x + 2)
```

Usa `\(x, y)` con funciones como `map2()` que tienen dos argumentos.

```
map2(l, p, \(x, y) x + y)
# se convierte
map2(l, p, function(l, p) l + p)
```

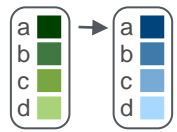
Use `\(x, y, z)`, etc. con funciones como `pmap()` que tienen muchos argumentos.

```
pmap(list(x, y, z), \(x, y, z) x + y / z)
# se convierte
pmap(list(x, y, z), function(x, y, z) x * (y + z))
```

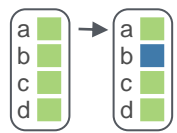
Usa `\(x, y)` con funciones como `imap()`. `x` obtendrá el valor de la lista y `y` obtendrá el índice, o el nombre si está disponible.

```
imap(list("a", "b", "c"), \(x, y) paste0(y, ":", x))
# salidas "index: value" para cada elemento
```

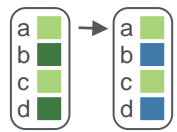
Modificar



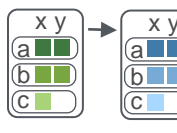
modify(.x, .f, ...) Aplique una función a cada elemento. Además, **modify2()**, y **imodify()**.
`modify(x, ~.+ 2)`



modify_at(.x, .at, .f, ...) Aplique una función a los elementos seleccionados. Además, **map_at()**.
`modify_at(x, "b", ~.+ 2)`



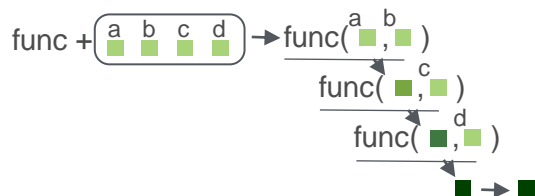
modify_if(.x, .p, .f, ...) Apply a function to elements that pass a test. Also **map_if()**.
`modify_if(x, is.numeric, ~.+ 2)`



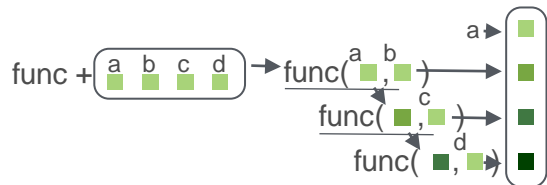
modify_depth(.x, .depth, .f, ...) Aplique la función a cada elemento en un nivel determinado de una lista. Además, **map_depth()**.
`modify_depth(x, 1, ~.+ 2)`

Reducir

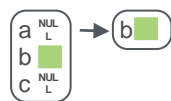
reduce(.x, .f, ..., .init, .dir = c("forward", "backward")) Aplique la función de forma recursiva a cada elemento de una lista o vector. Además, **reduce2()**.
`reduce(x, sum)`



accumulate(.x, .f, ..., .init) Reduzca una lista, pero también devuelva resultados intermedios. Además, **accumulate2()**.
`accumulate(x, sum)`



Vectores



compact(.x, .p = identity) Descarte los elementos vacíos.
`compact(x)`



keep_at(x, at) Conservar/descartar elementos según el nombre o la posición. En cambio **discard_at()**.
`keep_at(x, "a")`

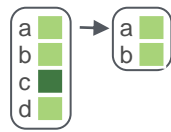


set_names(x, nm = x) Establezca los nombres de un vector/lista directamente o con una función.
`set_names(x, c("p", "q", "r"))`
`set_names(x, tolower)`

Funciones de predicado



keep(.x, .p, ...) Mantenga los elementos que pasan una prueba lógica. En cambio, **discard()**.
`keep(x, is.numeric)`



head_while(.x, .p, ...) Regrese los elementos de la cabeza hasta que uno no pase. Además, **tail_while()**.
`head_while(x, is.character)`



detect(.x, .f, ..., .dir = c("forward", "backward"), .right = NULL, .default = NULL) Encuentre el primer elemento para pasar. `detect(x, is.character)`



detect_index(.x, .f, ..., .dir = c("forward", "backward"), .right = NULL) Busque el índice del primer elemento que se va a pasar. `detect_index(x, is.character)`



every(.x, .p, ...) ¿Todos los elementos pasan una prueba?
`every(x, is.character)`



some(.x, .p, ...) ¿Algunos elementos pasan una prueba?
`some(x, is.character)`



none(.x, .p, ...) ¿Ningún elemento pasa una prueba?
`none(x, is.character)`



has_element(.x, .y) ¿Una lista contiene un elemento?
`has_element(x, "foo")`

Pluck



pluck(.x, ..., .default=NULL) Seleccione un elemento por nombre o índice. Además, **attr_getter()** y **chuck()**.
`pluck(x, "b")`
`x |> pluck("b")`

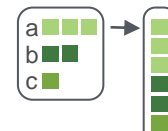


assign_in(x, where, value) Asigne un valor a una ubicación mediante la selección.
`assign_in(x, "b", 5)`
`x |> assign_in("b", 5)`



modify_in(.x, .where, .f) Aplicar una función a un valor en una ubicación seleccionada.
`modify_in(x, "b", abs)`
`x |> modify_in("b", abs)`

Reformar



list_flatten(.x) Eliminar un nivel de índices de una lista.
`list_flatten(x)`

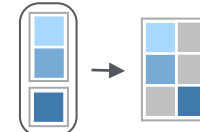
Concatenar



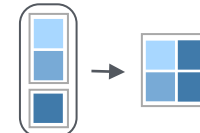
```
x1 <- list(a = 1, b = 2, c = 3)
x2 <- list(
  a = data.frame(x = 1:2),
  b = data.frame(y = "a")
)
```



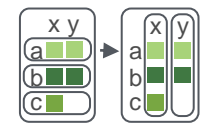
list_c(x) Combina elementos en un vector concatenándolos.
`list_c(x1)`



list_rbind(x) Combina elementos en un marco de datos mediante el enlace de filas.
`list_rbind(x2)`



list_cbind(x) Combina elementos en un marco de datos enlazándolos entre sí mediante la unión de columnas.
`list_cbind(x2)`



list_transpose(.l, .names = NULL) Transpone el orden de índice en una lista de varios niveles.
`list_transpose(x)`

Columnas-lista

Las columnas de lista son columnas de un marco de datos donde cada elemento es una lista o vector en lugar de un valor atómico. Las columnas también pueden ser listas de marcos de datos. Consulte tidy para obtener más información sobre los datos anidados y las columnas de lista.

TRABAJAR CON COLUMNAS DE LISTA

Manipule las columnas de la lista como cualquier otro tipo de columna, usando funciones dplyr como `mutate()`. Dado que cada elemento es una lista, utilice funciones de mapa dentro de una función de columna para manipular cada elemento.

map(), map2(), o pmap() devuelve listas y creará nuevas columnas-lista.

```
starwars |>
  transmute(ships = map2(vehicles,
    starships,
    append))
```

Las funciones de mapa con sufijo como `map_int()` devuelven un tipo de datos atómicos y simplificarán las columnas de lista en columnas regulares.

```
starwars |>
  mutate(n_films = map_int(films,
    length))
```