Apply functions with purrr :: CHEATSHEET

Map Functions

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

map(l1, sort, decreasing = TRUE)
l1 <- list(x = c("a", "b"), y = c("c", "d"))

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

map2(l, function(x) x + 2)

L1 <- list(a = 1:10, b = 11:20, c = 21:30)

OF A LIST OR VECTOR, AND RETURN A LIST.

map(l, sort, decreasing = TRUE)
l1 <- list(x = c("a", "b"), y = c("c", "d"))

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']])

FUNCTION SHORTCUTS

map(...)
map2(...)
imap(...)
pmap(...)
pwalk(...)

USE \
\{x\} WITH FUNCTIONS LIKE map() THAT HAVE SINGLE ARGUMENTS.

map(l, x + 2)
becomes
map(l, function(x) x + 2)

USE \{x, y\} WITH FUNCTIONS LIKE map2() THAT HAVE TWO ARGUMENTS.

map2(l, p, l + p)
becomes
map2(l, p, function(l, p) l + p)

USE \{x, y\}, \{x, y, z\} ETC WITH FUNCTIONS LIKE pmap() THAT HAVE MANY ARGUMENTS.

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

USE \{x\} 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"), \{x, y\} paste0(y, ": ", x))
outputs "index: value" for each item.
### 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, ~.+ 2)`
- **modify_depth(x, depth, f, ...)** Apply a function to each element at a given level of a list. Also map_depth().`modify_depth(x, 1, ~+ 2)`

### 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)`
- **accumulate(x, f, ..., .init)** Reduce a list, but also return intermediate results. Also accumulate2().`accumulate(x, sum)`

### Predicate functions

- **keep(x, p, ...)** Keep elements that pass a logical test. Conversely, **discard()**.`keep(x, is.numeric)`
- **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")`

### Pluck

- **pluck(x, ..., default=NULL)** Select an element by name or index. Also attr_getter() and **chuck()**.
- **assign_in(x, where, value)** Assign a value to a location using pluck selection. `assign_in(x, "b", 5)`
- **modify_in(x, .where, .f)** Apply a function to a value at a selected location. `modify_in(x, "b", abs)`

### Concatenate

- **list_c(x)** Combines elements into a vector by concatenating them together. `list_c(x1)`
- **list_rbind(x)** Combines elements into a data frame by row-binding them together. `list_rbind(x2)`
- **list_cbind(x)** Combines elements into a data frame by column-binding them together. `list_cbind(x2)`

### Reshape

- **list_flatten(x)** Remove a level of indexes from a list. `list_flatten(x)`
- **list_rtranspose(l, ..., names = NULL)** Transposes the index order in a multi-level list. `list_rtranspose(x)`

### List-Coordinates

- **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(). Because each element is a list, use map functions within a column function to manipulate each element.