Apply functions with purrr :: CHEAT SHEET

### Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

- `map(x, f, ...)` Apply a function to each element of a list or vector, `map(x, is.logical)`
- `map2(x, y, f, ...)` Apply a function to pairs of elements from two lists, vectors. `map2(x, y, sum)`
- `pmap(l, f, ...)` Apply a function to groups of elements from a list of lists, vectors. `pmap$lst(l, x, y, z)`
- `invoke_map(l, f, x = list(NULL, ..., env = NULL))` Run each function in a list. Also `invoke.l(lst, x = 1:9)`

### Work with Lists

**FILTER LISTS**

- `pluck(x, ...)` Select an element by name or index, `pluck(x, "b")`, or its attribute with `attr_getter`.
- `keep(x, p, ...)` Select elements that pass a logical test, `keep(x, is.na)`
- `discard(x, p, ...)` Select elements that do not pass a logical test, `discard(x, is.na)`
- `compact(x, p = identity)` Drop empty elements. `compact(x)`
- `head_while(x, p, ...)` Return head elements until one does not pass. `head_while(x, is.character)`
- `tail_while(x, p, ...)` Return tail elements until one does not pass. `tail_while(x, is.character)`
- `head(x)` Return first `n` elements.
- `tail(x)` Return last `n` elements.

**SUMMARISE LISTS**

- `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)`
- `has_element(x, y)` Does a list contain an element? `has_element(x, "foo")`
- `detect(x, f, ..., right = FALSE, .p)` Find first element to pass. `detect(x, is.character)`
- `detect_index(x, f, ..., right = FALSE, .p)` Find index of first element to pass. `detect_index(x, is.character)`
- `vec_depth(x)` Return depth (number of levels of indexes), `vec_depth(x)`

**RESHAPE LISTS**

- `flatten(x)` Remove a level of indexes from a list. Also `flatten_chr, flatten_dbl, flatten_dfc, flatten_dfr, flatten_int, flatten_lgl, flatten(x)`
- `transpose(l, ...)` Transposes the index order in a multi-level list. `transpose(x)`
- `array_tree(x, margin = 3)` Turn array into list. `array_tree(x, margin = 3)`
- `transmute(x, y)` Transforms the input invisibly.
- `modify(x, ~.+2)` Modify function behavior

**JOIN (TO) LISTS**

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

**WORK WITH LISTS**

- `array_tree(array, margin = 3)` Turn array into list. Also `array_branch, array_tree(x, margin = 3)`
- `cross(x, y, filter = NULL)` All combinations of x and y. Also `cross3, cross_dfr, cross2(1:3, 4:6)`
- `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, tolerow)`

### Reduce Lists

- `reduce(x, f, ..., .init)` Apply function recursively to each element of a list or vector. Also `reduce2, reduce2_right, reduce2.init, reduce(x, sum)`
- `accumulate(x, f, ..., .init)` Reduce, but also return intermediate results. Also `accumulate_right, accumulate(x, sum)`

**Modify function behavior**

- `compose()` Compose function multiple functions.
- `lift()` Change the type of input a function takes. Also `lift_dlv, lift_dv, lift_id, lift_ltv, lift_vlv`
- `rerun()` Rerun expression n times.
- `negate()` Negate a predicate function (a pipe friendly!)
- `quietly()` Modify function to return list of results, output, messages, warnings.
- `possibly()` Modify function to return default value whenever an error occurs (instead of error).
Nested Data

A nested data frame stores individual tables within the cells of a larger, organizing table.

Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the purrr functions map(), map2(), or pmap().

Use a two-step process to create a nested data frame:

1. Group the data frame into groups with setosa, setosa, and virginica species.
2. Use nest() to create a nested data frame with one row per group.

Unnest a nested data frame with unnest():

Use the purrr functions map_lgl(), map_int(), map_dbl(), and map_chr(), as well as tidy's unnest() to reduce a list column into a regular column.

List Column Workflow

1. Make a list column

   ```r
tibble::tribble()
   ``

   Makes list column when needed

   ```r
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
   ```

   Saves list input as list columns

   ```r
   tribble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
   ```

2. Work with list columns

   ```r
tibble::enframe()
   ``

   Converts multi-level list to tibble with list cols enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

   ```r
   tribble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
   ```

3. Simplify the list column

   ```r
tibble::transmute()
   ``

   Transforms a list column into a regular column.

   ```r
   transmute(model = map(data, fun))
   ```

   Appply a function that returns a result element-wise for list columns:

   ```r
   m_iris %=% transmute(model = map(data, fun))
   ```

Saves list input as list columns

```r
transmute(model = map(data, fun))
```