9_Functionals

2023-01-11

9. Functionals

9.1 Introduction

```
randomise <- function(f) f(runif(1e3))
randomise(mean)
```

```
## [1] 0.4916555
```

```
randomise(mean)
```

```
## [1] 0.4986182
```

```
randomise(sum)
```

```
## [1] 495.4548
```

```
library(purrr)
```

9.2 My first functional: map()

```
triple <- function(x) x * 3
map(1:3, triple)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 9
```

```
simple_map <- function(x, f, ...) {
  out <- vector("list", length(x))
  for (i in seq_along(x)) {
    out[[i]] <- f(x[[i]], ...)
  }
  out
}
```

9.2.1 Producing atomic vectors

```
map_chr(mtcars, typeof)
```

```
##      mpg      cyl     disp       hp     drat       wt     qsec       vs
## "double" "double" "double" "double" "double" "double" "double" "double"
##       am     gear     carb
## "double" "double" "double"
```

```
map_lgl(mtcars, is.double)
```

```
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
## TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
n_unique <- function(x) length(unique(x))
map_int(mtcars, n_unique)
```

```
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
##   25    3   27   22   22   29   30    2    2    3    6
```

```
map_dbl(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

```
pair <- function(x) c(x, x)
map_dbl(1:2, pair)
```

```
## Error in `map_dbl()`:
## i In index: 1.
## Caused by error:
## ! Result must be length 1, not 2.
```

```
map_dbl(1:2, as.character)
```

```
## Error in `map_dbl()`:
## i In index: 1.
## Caused by error:
## ! Can't coerce from a character vector to a double vector.
```

```
map(1:2, pair)
```

```
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 2 2
```

```r
map(1:2, as.character)
```

```
## [[1]]
## [1] "1"
##
## [[2]]
## [1] "2"
```

```r
map_dbl(x, mean, na.rm = TRUE)

=====

vapply(x, mean, na.rm = TRUE, FUN.VALUE = double(1))
```

```
## Error: <text>:3:1: unexpected '=='
## 2:
## 3: ==
##    ^
```

9.2.2 Anonymous functions and shortcuts

```r
map_dbl(mtcars, function(x) length(unique(x)))
```

```
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
##   25    3   27   22   22   29   30    2    2    3    6
```

```r
map_dbl(mtcars, ~ length(unique(.x)))
```

```
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
##   25    3   27   22   22   29   30    2    2    3    6
```

```r
as_mapper(~ length(unique(.x)))
```

```
## <lambda>
## function (..., .x = ..1, .y = ..2, . = ..1)
## length(unique(.x))
## attr(,"class")
## [1] "rlang_lambda_function" "function"
```

```r
x <- map(1:3, ~ runif(2))
str(x)
```

```
## List of 3
##  $ : num [1:2] 0.381 0.468
##  $ : num [1:2] 0.909 0.223
##  $ : num [1:2] 0.337 0.863
```

```r
x <- list(
  list(-1, x = 1, y = c(2), z = "a"),
  list(-2, x = 4, y = c(5, 6), z = "b"),
  list(-3, x = 8, y = c(9, 10, 11))
)
```

```r
map_dbl(x, "x")
```

```
## [1] 1 4 8
```

```r
map_dbl(x, 1)
```

```
## [1] -1 -2 -3
```

```r
map_dbl(x, list("y", 1))
```

```
## [1] 2 5 9
```

```r
map_chr(x, "z")
```

```
## Error in `map_chr()`:
## i In index: 3.
## Caused by error:
## ! Result must be length 1, not 0.
```

```r
map_chr(x, "z", .default = NA)
```

```
## [1] "a" "b" NA
```

9.2.3 Passing arguments with ...

```r
x <- list(1:5, c(1:10, NA))
map_dbl(x, ~ mean(.x, na.rm = TRUE))
```

```
## [1] 3.0 5.5
```

```r
map_dbl(x, mean, na.rm = TRUE)
```

```
## [1] 3.0 5.5
```

```r
plus <- function(x, y) x + y

x <- c(0, 0, 0, 0)
map_dbl(x, plus, runif(1))
```

```
## [1] 0.6460774 0.6460774 0.6460774 0.6460774
```

```r
map_dbl(x, ~ plus(.x, runif(1)))
```

```
## [1] 0.01802547 0.19352020 0.25556942 0.62215691
```

9.2.4 Argument names

```r
boostrap_summary <- function(x, f) {
  f(sample(x, replace = TRUE))
}

simple_map(mtcars, boostrap_summary, f = mean)
```

```
## Error in mean.default(x[[i]], ...): 'trim' must be numeric of length one
```

9.2.5 Varying another argument

```r
trims <- c(0, 0.1, 0.2, 0.5)
x <- rcauchy(1000)
```

```r
map_dbl(trims, ~ mean(x, trim = .x))
```

```
## [1]  0.33235079 -0.06352681 -0.06737572 -0.10324362
```

```r
map_dbl(trims, function(trim) mean(x, trim = trim))
```

```
## [1]  0.33235079 -0.06352681 -0.06737572 -0.10324362
```

```r
map_dbl(trims, mean, x = x)
```

```
## [1]  0.33235079 -0.06352681 -0.06737572 -0.10324362
```

9.2.6 Exercises

1. Use `as_mapper()` to explore how purrr generates anonymous functions for the integer, character, and list helpers. What helper allows you to extract attributes? Read the documentation to find out.

```r
as_mapper(c("a", "b", "c"))
```

```
## function (x, ...)
## pluck_raw(x, list("a", "b", "c"), .default = NULL)
## <environment: 0x000001aef18ed3e8>
```

```r
as_mapper(c(1, 2, 3))
```

```
## function (x, ...)
## pluck_raw(x, list(1, 2, 3), .default = NULL)
## <environment: 0x000001aef1aa5438>
```

```r
as_mapper(list(1, "a", 2))
```

```
## function (x, ...)
## pluck_raw(x, list(1, "a", 2), .default = NULL)
## <environment: 0x000001aef1c13510>
```

```r
as_mapper(list(1, attr_getter("a")))
```

```
## function (x, ...)
## pluck_raw(x, list(1, function (x)
## attr(x, attr, exact = TRUE)), .default = NULL)
## <environment: 0x000001aef1d969c0>
```

Looks like it is using `pluck_raw`. Get attributes with `attr_getter`

2. `map(1:3, ~ runif(2))` is a useful pattern for generating random numbers, but `map(1:3, runif(2))` is not. Why not? Can you explain why it returns the result that it does?

```r
map(1:3, ~ runif(2))
```

```
## [[1]]
## [1] 0.63387781 0.04513207
##
## [[2]]
## [1] 0.01863754 0.92722358
##
## [[3]]
## [1] 0.113676379 0.005741677
```

```r
map(1:3, runif(2))
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```r
as_mapper(map(1:3, runif(2)))
```

```
## function (x, ...)
## pluck_raw(x, list(1L, 2L, 3L), .default = NULL)
## <environment: 0x000001aef2312bb8>
```

```r
as_mapper(runif(2))
```

```
## function (x, ...)
## pluck_raw(x, list(0.545706412522122, 0.439693666994572), .default = NULL)
## <environment: 0x000001aef251a890>
```

First one creates an anonymous function which then generates 2 random numbers for each of the 3 iterations. The second only generates one set of random numbers which is fed into map and results in maps default values being spit out when piped to `as_mapper`

3. Use the appropriate `map()` function to:

- Compute the standard deviation of every column in a numeric data frame

```r
mat <- as.data.frame(matrix(1:25, nrow = 5))
map_dbl(mat, ~ sd(.x))
```

```
##       V1       V2       V3       V4       V5
## 1.581139 1.581139 1.581139 1.581139 1.581139
```

- Compute the standard deviation of every numeric column in a mixed data frame. (Hint: you'll need to do it in two steps.)

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```r
map_dbl(iris[map_lgl(iris, is.numeric)], ~ sd(.x))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.8280661    0.4358663    1.7652982    0.7622377
```

- Compute the number of levels for every factor in a data frame

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v dplyr   1.0.10
## v tibble  3.1.8      v stringr 1.5.0
## v tidyr   1.2.1      v forcats 0.5.2
## v readr   2.1.3
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
summary(attenu)
```

```
##      event            mag           station         dist
##  Min.   : 1.00   Min.   :5.000   117    :  5   Min.   :  0.50
##  1st Qu.: 9.00   1st Qu.:5.300   1028   :  4   1st Qu.: 11.32
##  Median :18.00   Median :6.100   113    :  4   Median : 23.40
##  Mean   :14.74   Mean   :6.084   112    :  3   Mean   : 45.60
##  3rd Qu.:20.00   3rd Qu.:6.600   135    :  3   3rd Qu.: 47.55
##  Max.   :23.00   Max.   :7.700   (Other):147   Max.   :370.00
##                                  NA's   : 16
##      accel
##  Min.   :0.00300
##  1st Qu.:0.04425
##  Median :0.11300
##  Mean   :0.15422
##  3rd Qu.:0.21925
##  Max.   :0.81000
##
```

```
df <- attenu %>%
  mutate(event = as.factor(event))
summary(df)
```

```
##      event          mag           station         dist             accel
##  19     :38   Min.   :5.000   117    :  5   Min.   :  0.50   Min.   :0.00300
##  9      :22   1st Qu.:5.300   1028   :  4   1st Qu.: 11.32   1st Qu.:0.04425
##  23     :18   Median :6.100   113    :  4   Median : 23.40   Median :0.11300
##  20     :16   Mean   :6.084   112    :  3   Mean   : 45.60   Mean   :0.15422
##  5      :11   3rd Qu.:6.600   135    :  3   3rd Qu.: 47.55   3rd Qu.:0.21925
##  18     :11   Max.   :7.700   (Other):147   Max.   :370.00   Max.   :0.81000
##  (Other):66                   NA's   : 16
```

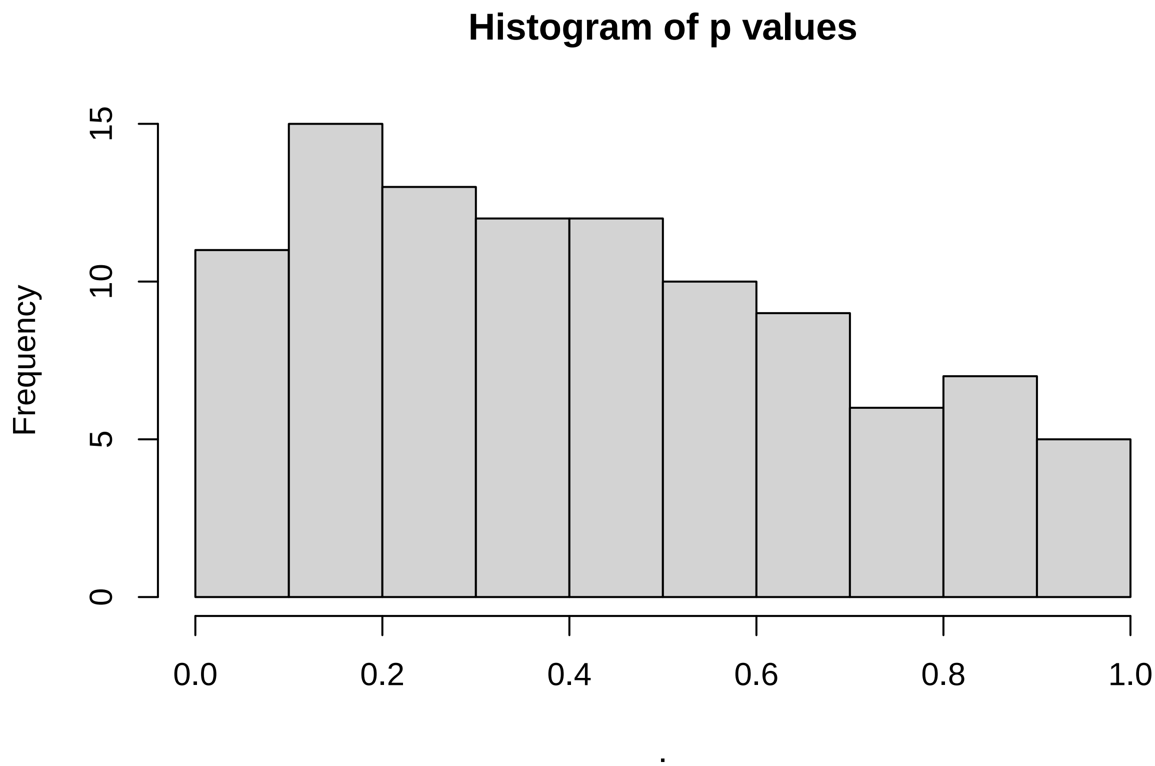
```
map_int(df[map_lgl(df, ~ is.factor(.x))], ~ length(levels(.x)))
```

```
##   event station
##      23     117
```

4. The following code simulates the performance of a t-test for non-normal data. Extract the p-value from each test, then visualise.

```
trials <- map(1:100, ~ t.test(rpois(10, 10), rpois(7, 10)))

map_dbl(trials, "p.value") %>%
  hist(main = "Histogram of p values")
```



5. The following code uses a map nested inside another map to apply a function to every element of a nested list. Why does it fail, and what do you need to do to make it work?

```
x <- list(
  list(1, c(3, 9)),
  list(c(3, 6), 7, c(4, 7, 6))
)

triple <- function(x) x * 3
map(x, map, .f = triple)
```

```
## Error in `map()`:
## i In index: 1.
## Caused by error in `.f()`:
## ! unused argument (function (.x, .f, ..., .progress = FALSE)
## {
##     map_("list", .x, .f, ..., .progress = .progress)
## })
```

```
map(x, map, triple)
```

```
## [[1]]
## [[1]][[1]]
## [1] 3
##
## [[1]][[2]]
## [1]  9 27
##
##
## [[2]]
## [[2]][[1]]
## [1]  9 18
##
## [[2]][[2]]
## [1] 21
##
## [[2]][[3]]
## [1] 12 21 18
```

```
# or
map(x, ~ map(.x, triple))
```

```
## [[1]]
## [[1]][[1]]
## [1] 3
##
## [[1]][[2]]
## [1]  9 27
##
##
## [[2]]
## [[2]][[1]]
## [1]  9 18
##
## [[2]][[2]]
## [1] 21
##
## [[2]][[3]]
## [1] 12 21 18
```

Using .f makes triple the function of the outer map call and not the inner map call. Just remove the name and it will go in order and work

6. Use `map()` to fit linear models to the mtcars dataset using the formulas stored in this list:

```
formulas <- list(
  mpg ~ disp,
  mpg ~ I(1 / disp),
  mpg ~ disp + wt,
  mpg ~ I(1 / disp) + wt
```

```r
)

map(formulas, lm, data = mtcars)
```

```
## [[1]]
##
## Call:
## .f(formula = .x[[i]], data = ..1)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
##
##
## [[2]]
##
## Call:
## .f(formula = .x[[i]], data = ..1)
##
## Coefficients:
## (Intercept)    I(1/disp)
##       10.75      1557.67
##
##
## [[3]]
##
## Call:
## .f(formula = .x[[i]], data = ..1)
##
## Coefficients:
## (Intercept)         disp           wt
##    34.96055     -0.01772     -3.35083
##
##
## [[4]]
##
## Call:
## .f(formula = .x[[i]], data = ..1)
##
## Coefficients:
## (Intercept)    I(1/disp)           wt
##      19.024     1142.560       -1.798
```

7. Fit the model `mpg ~ disp` to each of the bootstrap replicates of `mtcars` in the list below, then extract the R^2 of the model fit (Hint: you can compute the R^2 with summary().)

```r
bootstrap <- function(df) {
  df[sample(nrow(df), replace = TRUE), , drop = FALSE]
}

bootstraps <- map(1:10, ~ bootstrap(mtcars))
head(bootstraps)
```

```
## [[1]]
##                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 450SL           17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Pontiac Firebird     19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Hornet 4 Drive       21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Mazda RX4            21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Cadillac Fleetwood   10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Mazda RX4 Wag        21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Mazda RX4 Wag.1      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Valiant              18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Valiant.1            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Hornet Sportabout    18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Ford Pantera L       15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 240D            24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Toyota Corona        21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9            27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Fiat 128             32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla       33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128.1           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Valiant.2            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Porsche 914-2        26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Volvo 142E           21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Duster 360           14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial    14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Hornet Sportabout.1  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## AMC Javelin          15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Cadillac Fleetwood.1 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Toyota Corolla.1     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Volvo 142E.1         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Hornet 4 Drive.1     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Camaro Z28           13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Duster 360.1         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Ford Pantera L.1     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Fiat X1-9.1          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
##
## [[2]]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Fiat X1-9.1         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat 128.1          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Merc 280.1          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
## Hornet 4 Drive.1    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E.1        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Valiant.1           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Lotus Europa.1      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Merc 280.2          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Chrysler Imperial.1 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Merc 450SLC.1       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Hornet 4 Drive.2    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
##
## [[3]]
##                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 450SLC           15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Datsun 710            22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Honda Civic           30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lincoln Continental   10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Hornet Sportabout     18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## AMC Javelin           15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Hornet Sportabout.1   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Lotus Europa          30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ferrari Dino          19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Lincoln Continental.1 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Honda Civic.1         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Hornet Sportabout.2   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360            14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Hornet Sportabout.3   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Porsche 914-2         26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Dodge Challenger      15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Fiat X1-9             27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Fiat 128              32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Cadillac Fleetwood    10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Camaro Z28            13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Mazda RX4             21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Maserati Bora         15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Toyota Corolla        33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Chrysler Imperial     14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora.1       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Chrysler Imperial.1   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Merc 240D             24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Lotus Europa.1        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Lotus Europa.2        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Merc 280C             17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Hornet 4 Drive        21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet 4 Drive.1      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
##
## [[4]]
##                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## AMC Javelin           15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
```

```
## Merc 240D            24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Toyota Corolla       33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Hornet 4 Drive       21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 450SL           17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Fiat X1-9            27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Ferrari Dino         19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Merc 450SLC          15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Datsun 710           22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 450SE           16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Duster 360           14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Toyota Corona        21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Toyota Corona.1      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Merc 280C            17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Lincoln Continental  10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Dodge Challenger     15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Camaro Z28           13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Hornet 4 Drive.1     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 280C.1          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Lincoln Continental.1 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Ford Pantera L       15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 280C.2          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Valiant              18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280C.3          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 280             19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Volvo 142E           21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Dodge Challenger.1   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Camaro Z28.1         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Mazda RX4 Wag        21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet Sportabout    18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Porsche 914-2        26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Camaro Z28.2         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
##
## [[5]]
##                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat X1-9            27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Toyota Corona        21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Pontiac Firebird     19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Merc 280C            17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Lotus Europa         30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Merc 450SL           17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Hornet 4 Drive       21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 230             22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 450SL.1         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Dodge Challenger     15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Cadillac Fleetwood   10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Camaro Z28           13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Merc 240D            24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Hornet Sportabout    18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Datsun 710           22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Porsche 914-2        26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Merc 280C.1          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Honda Civic          30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa.1       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Hornet Sportabout.1  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
```

```
## Mazda RX4 Wag        21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Merc 240D.1          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Ferrari Dino         19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Merc 280             19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## AMC Javelin          15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Merc 450SLC          15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Honda Civic.1        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Volvo 142E           21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Lincoln Continental  10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Dodge Challenger.1   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Cadillac Fleetwood.1 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lotus Europa.2       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
##
## [[6]]
##                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 280C             17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Mazda RX4 Wag         21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Mazda RX4 Wag.1       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Fiat X1-9             27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Valiant               18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Lotus Europa          30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Maserati Bora         15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Datsun 710            22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Chrysler Imperial     14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Mazda RX4             21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Merc 280C.1           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Lincoln Continental   10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Hornet Sportabout     18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Datsun 710.1          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Fiat X1-9.1           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Chrysler Imperial.1   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Merc 280              19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Hornet 4 Drive        21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Lotus Europa.1        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Merc 240D             24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Honda Civic           30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Datsun 710.2          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Fiat X1-9.2           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Chrysler Imperial.2   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Merc 280.1            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Fiat 128              32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Merc 230              22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Lincoln Continental.1 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Maserati Bora.1       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 240D.1           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 450SL            17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Maserati Bora.2       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```
map(bootstraps, ~ lm(mpg ~ disp, data = .x)) %>%
  map(summary) %>%
  map("r.squared")
```

```
## [[1]]
## [1] 0.7490665
```

```
##
## [[2]]
## [1] 0.72722
##
## [[3]]
## [1] 0.8048088
##
## [[4]]
## [1] 0.6999852
##
## [[5]]
## [1] 0.7307035
##
## [[6]]
## [1] 0.7409532
##
## [[7]]
## [1] 0.7167109
##
## [[8]]
## [1] 0.6766594
##
## [[9]]
## [1] 0.7389807
##
## [[10]]
## [1] 0.7736669
```

9.3 Purrr style

```
by_cyl <- split(mtcars, mtcars$cyl)
```

```
by_cyl %>%
  map(~ lm(mpg ~ wt, data = .x)) %>%
  map(coef) %>%
  map_dbl(2)
```

```
##         4         6         8
## -5.647025 -2.780106 -2.192438
```

```
by_cyl %>%
  lapply(function(data) lm(mpg ~ wt, data = data)) %>%
  lapply(coef) %>%
  vapply(function(x) x[[2]], double(1))
```

```
##         4         6         8
## -5.647025 -2.780106 -2.192438
```

```
models <- lapply(by_cyl, function(data) lm(mpg ~ wt, data = data))
vapply(models, function(x) coef(x)[[2]], double(1))
```

```
##         4         6         8
## -5.647025 -2.780106 -2.192438
```

```
slopes <- double(length(by_cyl))
for (i in seq_along(by_cyl)) {
  model <- lm(mpg ~ wt, data = by_cyl[[i]])
  slopes[[i]] <- coef(model)[[2]]
}
slopes
```

```
## [1] -5.647025 -2.780106 -2.192438
```

9.4 Map variants

	List	Atomic	Same type	Nothing
One argument	`map()`	`map_lgl()`, …	`modify()`	`walk()`
Two arguments	`map2()`	`map2_lgl()`, …	`modify2()`	`walk2()`
One argument + index	`imap()`	`imap_lgl()`, …	`imodify()`	`iwalk()`
N arguments	`pmap()`	`pmap_lgl()`, …	—	`pwalk()`

9.4.1 Same type of output as input: `modify()`

```
df <- data.frame(
  x = 1:3,
  y = 6:4
)

map(df, ~ .x * 2)
```

```
## $x
## [1] 2 4 6
##
## $y
## [1] 12 10  8
```

```
modify(df, ~ .x * 2)
```

```
##   x  y
## 1 2 12
## 2 4 10
## 3 6  8
```

```r
df <- modify(df, ~ .x * 2)
df
```

```
##   x  y
## 1 2 12
## 2 4 10
## 3 6  8
```

```r
simple_modify <- function(x, f, ...) {
  for (i in seq_along(x)) {
    x[[i]] <- f(x[[i]], ...)
  }
  x
}
```

```r
modify_if(df, is.numeric, ~ .x * 2)
```

```
##    x  y
## 1  4 24
## 2  8 20
## 3 12 16
```

9.4.2 Two inputs: map2() and friends

```r
xs <- map(1:8, ~ runif(10))
xs[[1]][[1]] <- NA
ws <- map(1:8, ~ rpois(10, 5) + 1)
```

```r
map_dbl(xs, mean)
```

```
## [1]        NA 0.5220627 0.4445931 0.4726293 0.4645845 0.4301546 0.3900846
## [8] 0.4990225
```

```r
map_dbl(xs, weighted.mean, w = ws)
```

```
## Error in `map_dbl()`:
## i In index: 1.
## Caused by error in `weighted.mean.default()`:
## ! 'x' and 'w' must have the same length
```

```r
map2_dbl(xs, ws, weighted.mean)
```

```
## [1]        NA 0.5184438 0.4166630 0.4524956 0.4394998 0.4070773 0.4114907
## [8] 0.4825054
```

```r
map2_dbl(xs, ws, weighted.mean, na.rm = TRUE)
```

```
## [1] 0.5181299 0.5184438 0.4166630 0.4524956 0.4394998 0.4070773 0.4114907
## [8] 0.4825054
```

```
simple_map2 <- function(x, y, f, ...) {
  out <- vector("list", length(x))
  for (i in seq_along(x)) {
    out[[i]] <- f(x[[i]], y[[i]], ...)
  }
  out
}
```

9.4.3 No outputs: `walk()` and friends

```
welcome <- function(x) {
  cat("Welcome ", x, "!\n", sep = "")
}
names <- c("Hadley", "Jenny")

map(names, welcome)
```

```
## Welcome Hadley!
## Welcome Jenny!
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

```
walk(names, welcome)
```

```
## Welcome Hadley!
## Welcome Jenny!
```

```
temp <- tempfile()
dir.create(temp)

cyls <- split(mtcars, mtcars$cyl)
paths <- file.path(temp, paste0("cyl-", names(cyls), ".csv"))
walk2(cyls, paths, write.csv)

dir(temp)
```

```
## [1] "cyl-4.csv" "cyl-6.csv" "cyl-8.csv"
```

9.4.4 Iterating over values and indices

```
imap_chr(iris, ~ paste0("The first value of ", .y, " is ", .x[[1]]))
```

```
##                                Sepal.Length
## "The first value of Sepal.Length is 5.1"
##                                 Sepal.Width
##  "The first value of Sepal.Width is 3.5"
##                                Petal.Length
## "The first value of Petal.Length is 1.4"
##                                 Petal.Width
##  "The first value of Petal.Width is 0.2"
##                                     Species
##   "The first value of Species is setosa"
```

```
x <- map(1:6, ~ sample(1000, 10))
imap_chr(x, ~ paste0("The highest value of ", .y, " is ", max(.x)))
```

```
## [1] "The highest value of 1 is 923" "The highest value of 2 is 971"
## [3] "The highest value of 3 is 910" "The highest value of 4 is 926"
## [5] "The highest value of 5 is 892" "The highest value of 6 is 867"
```

9.4.5 Any number of inputs: pmap() and friends

```
pmap_dbl(list(xs, ws), weighted.mean)
```

```
## [1]        NA 0.5184438 0.4166630 0.4524956 0.4394998 0.4070773 0.4114907
## [8] 0.4825054
```

```
pmap_dbl(list(xs, ws), weighted.mean, na.rm = T)
```

```
## [1] 0.5181299 0.5184438 0.4166630 0.4524956 0.4394998 0.4070773 0.4114907
## [8] 0.4825054
```

```
trims <- c(0, 0.1, 0.2, 0.5)
x <- rcauchy(1000)

pmap_dbl(list(trim = trims), mean, x = x)
```

```
## [1]  0.80675708 -0.05975252 -0.03565473 -0.01025761
```

```
params <- tibble::tribble(
  ~ n, ~ min, ~ max,
   1L,     0,     1,
   2L,    10,   100,
   3L,   100,  1000
)

# runif(n, min, max)
pmap(params, runif)
```

```
## [[1]]
## [1] 0.1445937
##
## [[2]]
## [1] 14.52192 58.98318
##
## [[3]]
## [1] 353.5616 157.8975 236.1777
```

9.4.6 Exercises

1. Explain the results of `modify(mtcars, 1)`

```r
modify(mtcars, 1)
```

```
##    mpg cyl disp  hp drat   wt  qsec vs am gear carb
## 1   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 2   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 3   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 4   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 5   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 6   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 7   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 8   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 9   21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 10  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 11  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 12  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 13  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 14  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 15  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 16  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 17  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 18  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 19  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 20  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 21  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 22  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 23  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 24  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 25  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 26  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 27  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 28  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 29  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 30  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 31  21   6  160 110  3.9 2.62 16.46  0  1    4    4
## 32  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

The rows are all the same. The call extracts the first row of the `mtcars` data frame. Then since modify returns the same size output as input, it just recycles it for the original length of `mtcars`

2. Rewrite the following code to use `iwalk()` instead of `walk2()`. What are the advantages and disadvantages?

```
cyls <- split(mtcars, mtcars$cyl)
paths <- file.path(temp, paste0("cyl-", names(cyls), ".csv"))
walk2(cyls, paths, write.csv)

names(cyls) <- paths
iwalk(cyls, ~write.csv(.x, .y))
```

3. Explain how the following code transforms a data frame using functions stored in a list.

```
trans <- list(
  disp = function(x) x * 0.0163871,
  am = function(x) factor(x, labels = c("auto", "manual"))
)

nm <- names(trans)
mtcars[nm] <- map2(trans, mtcars[nm], function(f, var) f(var))
```

Compare and contrast the map2() approach to this map() approach:

```
mtcars[nm] <- map(nm, ~ trans[[.x]](mtcars[[.x]]))
```

The two functions in the list are disp for displacement which calculates displacement with a fixed value and am which converts the column to a factor column of either auto or manual for levels. nm is the names of the functions. The function them modifies the the two columns of mtcars which match the names of the two functions by iterating over similar indexes. In this case f would be our x and var would be our y. The second directly iterates over the names contained in the nm object

4. What does `write.csv()` return, i.e. what happens if you use it with `map2()` instead of `walk2()`

```
cyls <- split(mtcars, mtcars$cyl)
paths <- file.path(temp, paste0("cyl-", names(cyls), ".csv"))
walk2(cyls, paths, write.csv)
map2(cyls, paths, write.csv)
```

```
## $`4`
## NULL
##
## $`6`
## NULL
##
## $`8`
## NULL
```

It returns a list of length 3 with NULL as the return value which we don't care about.

9.5 Reduce family

9.5.1 Basics

```
l <- map(1:4, ~ sample(1:10, 15, replace = T))
str(l)
```

```
## List of 4
##  $ : int [1:15] 9 5 9 9 1 4 5 8 6 10 ...
##  $ : int [1:15] 9 10 5 8 9 10 7 6 9 3 ...
##  $ : int [1:15] 7 5 4 9 2 8 6 2 6 9 ...
##  $ : int [1:15] 5 4 2 3 10 4 4 9 2 6 ...
```

```
out <- l[[1]]
out <- intersect(out, l[[2]])
out <- intersect(out, l[[3]])
out <- intersect(out, l[[4]])
out
```

```
## [1] 9 5 4 8 6
```

```
reduce(l, intersect)
```

```
## [1] 9 5 4 8 6
```

```
reduce(l, union)
```

```
##  [1]  9  5  1  4  8  6 10  7  3  2
```

```
simple_reduce <- function(x, f) {
  out <- x[[1]]
  for (i in seq(2, length(x))) {
    out <- f(out, x[[i]])
  }
  out
}
```

9.5.2 Accumulate

```
accumulate(l, intersect)
```

```
## [[1]]
##  [1]  9  5  9  9  1  4  5  8  6 10  7  1  6  3  3
##
## [[2]]
## [1]  9  5  4  8  6 10  7  3
##
## [[3]]
```

```
## [1] 9 5 4 8 6 7
##
## [[4]]
## [1] 9 5 4 8 6
```

```
x <- c(4, 3, 10)
reduce(x, `+`)
```

```
## [1] 17
```

```
accumulate(x, `+`)
```

```
## [1]  4  7 17
```

9.5.3 Output types

```
reduce(1, `+`)
```

```
## [1] 1
```

```
reduce("a", `+`)
```

```
## [1] "a"
```

```
reduce(integer(), `+`)
```

```
## Error in `reduce()`:
## ! Must supply `.init` when `.x` is empty.
```

```
reduce(integer(), `+`, .init = 0)
```

```
## [1] 0
```

```
reduce("a", `+`, .init = 0)
```

```
## Error in .x + .y: non-numeric argument to binary operator
```

```
sum(integer())  # x + 0 = x
```

```
## [1] 0
```

```
prod(integer()) # x * 1 = x
```

```
## [1] 1
```

```r
min(integer())  # min(x, Inf) = x
```

```
## Warning in min(integer()): no non-missing arguments to min; returning Inf

## [1] Inf
```

```r
max(integer())  # max(x, -Inf) = x
```

```
## Warning in max(integer()): no non-missing arguments to max; returning -Inf

## [1] -Inf
```

9.5.4 Multiple inputs

9.5.5 Map-reduce

9.6 Predicate functionals

9.6.1 Basics

```r
df <- data.frame(x = 1:3, y = c("a", "b", "c"))
detect(df, is.factor)
```

```
## NULL
```

```r
detect_index(df, is.factor)
```

```
## [1] 0
```

```r
str(keep(df, is.factor))
```

```
## 'data.frame':    3 obs. of  0 variables
```

```r
str(discard(df, is.factor))
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

9.6.2 Map variants

```r
df <- data.frame(
  num1 = c(0, 10, 20),
  num2 = c(5, 6, 7),
  chr1 = c("a", "b", "c"),
  stringsAsFactors = FALSE
)
df
```

```
##   num1 num2 chr1
## 1    0    5    a
## 2   10    6    b
## 3   20    7    c
```

```r
str(map_if(df, is.numeric, mean))
```

```
## List of 3
##  $ num1: num 10
##  $ num2: num 6
##  $ chr1: chr [1:3] "a" "b" "c"
```

```r
str(modify_if(df, is.numeric, mean))
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ num1: num  10 10 10
##  $ num2: num  6 6 6
##  $ chr1: chr  "a" "b" "c"
```

```r
str(map(keep(df, is.numeric), mean))
```

```
## List of 2
##  $ num1: num 10
##  $ num2: num 6
```

9.6.3 Exercises

1. Why isn't `is.na()` a predicate function? What base R function is closest to being a predicate version of `is.na()`?

```r
is.na(1)
```

```
## [1] FALSE
```

```r
is.na(c(1,2))
```

```
## [1] FALSE FALSE
```

```r
is.na(c(NA,1,2))
```

```
## [1]  TRUE FALSE FALSE
```

```r
anyNA(1)
```

```
## [1] FALSE
```

```r
anyNA(c(1,2))
```

```
## [1] FALSE
```

```r
anyNA(c(1,2,NA))
```

```
## [1] TRUE
```

`is.na())` is not a predicate function because it returns a T or F for each element in the vector rather than just a single T or F. The closest base R function is `anyNA()`

2. `simple_reduce()` has a problem when x is length 0 or length 1. Describe the source of the problem and how you might go about fixing it.

```r
# Original
simple_reduce <- function(x, f) {
  out <- x[[1]]
  for (i in seq(2, length(x))) {
    out <- f(out, x[[i]])
  }
  out
}

simple_reduce(c(1,2,3), `+`)
```

```
## [1] 6
```

```r
simple_reduce(c(1), `+`)
```

```
## Error in x[[i]]: subscript out of bounds
```

```r
new_simple_reduce <- function(x, f) {
  if(length(x) < 2) return(x)
  out <- x[[1]]
  for (i in seq(2, length(x))) {
    out <- f(out, x[[i]])
  }
  out
}

new_simple_reduce(c(1,2,3), `+`)
```

```
## [1] 6
```

```r
new_simple_reduce(c(1), `+`)
```

```
## [1] 1
```

```r
new_simple_reduce(c(), `+`)
```

```
## NULL
```

`simple_reduce()` uses indexing which is hard coded to expect at least a length of 2. This can be fixed by checking the length of x before the for loop and assigning out.

3. Implement the `span()` function from Haskell: given a list `x` and a predicate function `f`, `span(x, f)` returns the location of the longest sequential run of elements where the predicate is true. (Hint: you might find `rle()` helpful.)

```r
x <- map(1:4, ~ sample(1:10, 10, replace = T))
f <- function(l) l %% 2 == 0
str(x)
```

```
## List of 4
##  $ : int [1:10] 6 2 10 5 1 6 4 3 7 10
##  $ : int [1:10] 10 1 6 4 1 3 9 5 10 2
##  $ : int [1:10] 8 5 2 3 3 7 8 2 4 8
##  $ : int [1:10] 3 10 1 6 5 10 1 5 6 1
```

```r
span <- function(x, f) {
  logic_x <- map(x, f)
  rle_x <- map(logic_x, rle)
  rle_x <- set_names(rle_x, nm = seq_along(rle_x))
  max_x <- map(rle_x, ~ max(.x[["lengths"]][.x[["values"]] == T]))
  idx_x <-
    map2(rle_x, max_x, ~ which(.x[["lengths"]] == .y &
                                .x[["values"]] == T))
  positions_x <- map2(rle_x, idx_x,
                      function(x, y) {
                        map(y, function(y2) {
                          if (y2 == 1) {
                            y2
                          } else {
                            sum(unlist(x[["lengths"]][1:(y2 - 1)])) + 1
                          }
                        })
                      })
  positions_x <- map(positions_x, unlist)
  return(positions_x)
}

span(x,f)
```

```
## $`1`
## [1] 1
##
## $`2`
## [1] 3 9
##
```

```
## $`3`
## [1] 7
##
## $`4`
## [1] 2 4 6 9
```

```r
span2 <- function(x, f) {
  logic_x <- map(x, f)
  rle_x <- map(logic_x, rle)
  rle_x <- set_names(rle_x, nm = seq_along(rle_x))
  maxes_x <- map(rle_x, ~ max(.x[["lengths"]][.x[["values"]] == T]))
  max_x <- reduce(map(rle_x, ~ max(.x[["lengths"]][.x[["values"]] == T])), max)
  max_idx <- which(maxes_x == max_x)
  filtered_rle_x <- rle_x[max_idx]
  idx_x <-
    map2(filtered_rle_x, max_x, ~ which(.x[["lengths"]] == .y &
                                  .x[["values"]] == T))
  positions_x <- map2(filtered_rle_x, idx_x,
                      function(x, y) {
                        map(y, function(y2) {
                          if (y2 == 1) {
                            y2
                          } else {
                            sum(unlist(x[["lengths"]][1:(y2 - 1)])) + 1
                          }
                        })
                      })
  positions_x <- map(positions_x, unlist)
  return(positions_x)
}
span2(x,f)
```

```
## $`3`
## [1] 7
```

4. Implement `arg_max()`. It should take a function and a vector of inputs, and return the elements of the input where the function returns the highest value. For example, `arg_max(-10:5, function(x) x ^ 2)` should return -10. `arg_max(-5:5, function(x) x ^ 2)` should return `c(-5, 5)`. Also implement the matching arg_min() function.

```r
arg_max <- function(x, f){
  values <- map_dbl(x, f)
  x[which(values == max(values))]
}

arg_max(-10:5, function(x) x ^ 2) # -10
```

```
## [1] -10
```

```r
arg_max(-5:5, function(x) x ^ 2) # c(-5,5)
```

```
## [1] -5  5
```

```r
arg_min <- function(x, f){
  values <- map_dbl(x, f)
  x[which(values == min(values))]
}

arg_min(-10:5, function(x) x ^ 2) # 0
```

```
## [1] 0
```

```r
arg_min(-5:5, function(x) x ^ 2) # 0
```

```
## [1] 0
```

5. The function below scales a vector so it falls in the range [0, 1]. How would you apply it to every column of a data frame? How would you apply it to every numeric column in a data frame?

```r
scale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
numeric_iris <- iris[, 1:4]
summary(numeric_iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

```r
head(numeric_iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

```r
#On all columns, error if not all numeric
head(modify(numeric_iris, scale01))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1   0.22222222   0.6250000   0.06779661  0.04166667
## 2   0.16666667   0.4166667   0.06779661  0.04166667
## 3   0.11111111   0.5000000   0.05084746  0.04166667
## 4   0.08333333   0.4583333   0.08474576  0.04166667
## 5   0.19444444   0.6666667   0.06779661  0.04166667
## 6   0.30555556   0.7916667   0.11864407  0.12500000
```

```r
#On all numeric columns
head(modify_if(iris, is.numeric, scale01))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1   0.22222222   0.6250000   0.06779661  0.04166667  setosa
## 2   0.16666667   0.4166667   0.06779661  0.04166667  setosa
## 3   0.11111111   0.5000000   0.05084746  0.04166667  setosa
## 4   0.08333333   0.4583333   0.08474576  0.04166667  setosa
## 5   0.19444444   0.6666667   0.06779661  0.04166667  setosa
## 6   0.30555556   0.7916667   0.11864407  0.12500000  setosa
```

9.7 Base functionals

9.7.1 Matrices and arrays

```r
apply(X, # the matrix or array to summarise.
      MARGIN, # an integer vector giving the dimensions to summarise over, 1 = rows, 2 = columns
      FUN, # a summary function
      ... # other arguments passed on to FUN
)
```

```r
a2d <- matrix(1:20, nrow = 5)
a2d
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```r
apply(a2d, 1, mean)
```

```
## [1]  8.5  9.5 10.5 11.5 12.5
```

```r
apply(a2d, 2, mean)
```

```
## [1]  3  8 13 18
```

```r
a3d <- array(1:24, c(2, 3, 4))
a3d
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
```

```
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

```
apply(a3d, 1, mean)
```

```
## [1] 12 13
```

```
apply(a3d, c(1, 2), mean)
```

```
##      [,1] [,2] [,3]
## [1,]   10   12   14
## [2,]   11   13   15
```

```
a1 <- apply(a2d, 1, identity)
identical(a2d, a1)
```

```
## [1] FALSE
```

```
a2 <- apply(a2d, 2, identity)
identical(a2d, a2)
```

```
## [1] TRUE
```

```
df <- data.frame(x = 1:3, y = c("a", "b", "c"))
apply(df, 2, mean)
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA

##  x  y
## NA NA
```

9.7.2 Mathematical concerns

```
integrate(sin, 0, pi)
```

```
## 2 with absolute error < 2.2e-14
```

```
str(uniroot(sin, pi * c(1 / 2, 3 / 2)))
```

```
## List of 5
##  $ root      : num 3.14
##  $ f.root    : num 1.22e-16
##  $ iter      : int 2
##  $ init.it   : int NA
##  $ estim.prec: num 6.1e-05
```

```
str(optimise(sin, c(0, 2 * pi)))
```

```
## List of 2
##  $ minimum  : num 4.71
##  $ objective: num -1
```

```
str(optimise(sin, c(0, pi), maximum = TRUE))
```

```
## List of 2
##  $ maximum  : num 1.57
##  $ objective: num 1
```

9.7.3 Exercises

1. How does `apply()` arrange the output? Read the documentation and perform some experiments.

```
a2d <- matrix(1:20, nrow = 5)
rownames(a2d) <- paste0("Row", 1:5)
colnames(a2d) <- paste0("Col", 1:4)
a2d
```

```
##      Col1 Col2 Col3 Col4
## Row1    1    6   11   16
## Row2    2    7   12   17
## Row3    3    8   13   18
## Row4    4    9   14   19
## Row5    5   10   15   20
```

```
apply(a2d, 1, identity)
```

```
##      Row1 Row2 Row3 Row4 Row5
## Col1    1    2    3    4    5
## Col2    6    7    8    9   10
## Col3   11   12   13   14   15
## Col4   16   17   18   19   20
```

```
apply(a2d, 2, identity)
```

```
##      Col1 Col2 Col3 Col4
## Row1    1    6   11   16
## Row2    2    7   12   17
## Row3    3    8   13   18
## Row4    4    9   14   19
## Row5    5   10   15   20
```

```
# simplify on by default
### c(n, dim(X)[MARGIN])  if n > 1
### vector if n == 1
```

It works on the margin being operated on. In the first case it fills in across the row and there's 5 rows so it then creates 5 columns. In the second cause the margin is column so it files in by column and there are 4 columns so it fills in 4 columns.

2 What do `eapply()` and `rapply()` do? Does purrr have equivalents?

```
# eapply applies a function over values in an environment

env <- new.env(hash = FALSE) # so the order is fixed
env$a <- 1:10
env$beta <- exp(-3:3)
env$logic <- c(TRUE, FALSE, FALSE, TRUE)
# what have we there?
utils::ls.str(env)
```

```
## a :  int [1:10] 1 2 3 4 5 6 7 8 9 10
## beta :  num [1:7] 0.0498 0.1353 0.3679 1 2.7183 ...
## logic :  logi [1:4] TRUE FALSE FALSE TRUE
```

```
# compute the mean for each list element
       eapply(env, mean)
```

```
## $logic
## [1] 0.5
##
## $beta
## [1] 4.535125
##
## $a
## [1] 5.5
```

```
#rapply is a to recursively apply a function to a list
X <- list(list(a = pi, b = list(c = 1L)), d = "a test")

rapply(X, function(x) x, how = "replace") -> X.; stopifnot(identical(X, X.))
rapply(X, sqrt, classes = "numeric", how = "replace") # only operates on numeric elements
```

```
## [[1]]
```

```
## [[1]]$a
## [1] 1.772454
##
## [[1]]$b
## [[1]]$b$c
## [1] 1
##
##
##
## $d
## [1] "a test"
```

```r
X <- list(list(a = pi, b = list(c = c(1L, 2L, 3L), c2 = 2L)), d = "a test", e = c(1,2,3,4,5))
rapply(X, sqrt, classes = "numeric", how = "replace")
```

```
## [[1]]
## [[1]]$a
## [1] 1.772454
##
## [[1]]$b
## [[1]]$b$c
## [1] 1 2 3
##
## [[1]]$b$c2
## [1] 2
##
##
##
## $d
## [1] "a test"
##
## $e
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```r
rapply(X, sqrt, classes = "numeric", how = "unlist")
```

```
##        a       e1       e2       e3       e4       e5
## 1.772454 1.000000 1.414214 1.732051 2.000000 2.236068
```

closest is `modify_depth()`

3. Challenge: read about the fixed point algorithm. Complete the exercises using R.

Does not exist