

Functions: Part 1

Daniel Anderson

Week 6, Class 1

Agenda

- Everything is a function
- Components of a function
- Function workflows

Learning objectives

- Understand and be able to fluently refer to the three fundamental components of a function
- Understand the workflows that often lead to writing functions, and how you iterate from interactive work to writing a function
- Be able to write a few basic functions

Functions

Anything that carries out an operation in R is a function. For example

```
3 + 5
```

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

The `+` is a function (what's referred to as an *infix* function).

Any ideas on how we could re-write the above to make it look more "function"-y?

```
`+`(3, 5)
```

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

What about this?

```
3 + 5 + 7
```

```
## [1] 15
```

```
`+`(7, `+`(3, 5))
```

```
## [1] 15
```

or

```
library(magrittr)  
`+`(3, 5) %>%  
  `+`(7)
```

```
## [1] 15
```

What's going on here?

- The `+` operator is a function that takes two arguments (both numeric), which it sums.
- The following are also the same (minus what's being assigned)

```
a <- 7  
a
```

```
## [1] 7
```

```
`<-`(a, 5)  
a
```

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

Everything is a function!

Being devious

- Want to introduce a devious bug? Redefine `+`

```
`+` <- function(x, y) {  
  if(runif(1) < 0.01) {  
    sum(x, y) * -1  
  } else {  
    sum(x, y)  
  }  
}  
table(map2_dbl(1:500, 1:500, `+`) > 0)
```

```
##  
## FALSE TRUE  
##      5   495
```

```
rm(`+`, envir = globalenv())  
table(map2_dbl(1:500, 1:500, `+`) > 0)
```

```
##  
## FALSE TRUE  
##      6   494
```

Tricky...

Functions are also (usually) objects!

```
a <- lm  
a(hp ~ drat + wt, data = mtcars)
```

```
##  
## Call:  
## a(formula = hp ~ drat + wt, data = mtcars)  
##  
## Coefficients:  
## (Intercept)          drat           wt  
##      -27.782         5.354        48.244
```


What does this all mean?

- Anything that carries out ANY operation in R is a function
- Functions are generally, but not always, stored in an object (otherwise known as binding the function to a name)

Anonymous functions

- The function for computing the mean is bound the name `mean`
- When running things through loops, you may often want to apply a function without binding it to a name

Example

```
vapply(mtcars, function(x) length(unique(x)), FUN.VALUE = double
```

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

Another possibility

- If you have a bunch of functions, you might consider storing them all in a list.
- You can then access the functions in the same way you would subset any list

```
funcs <- list(  
  quarter = function(x) x / 4,  
  half = function(x) x / 2,  
  double = function(x) x * 2,  
  quadruple = function(x) x * 4  
)
```

This is kind of weird...

```
funcs$quarter(100)
```

```
## [1] 25
```

```
funcs[["half"]](100)
```

```
## [1] 50
```

```
funcs[[4]](100)
```

```
## [1] 400
```

What does this imply?

- If we can store functions in a vector (list), then we can loop through the vector just like any other!

```
smry <- list(n = length,  
             n_miss = function(x) sum(is.na(x)),  
             n_valid = function(x) sum(!is.na(x)),  
             mean = mean,  
             sd = sd)
```

```
map_dbl(smry, ~.x(mtcars$mpg))
```

```
##           n      n_miss  n_valid      mean      sd
## 32.000000  0.000000 32.000000 20.090625  6.026948
```

Careful though

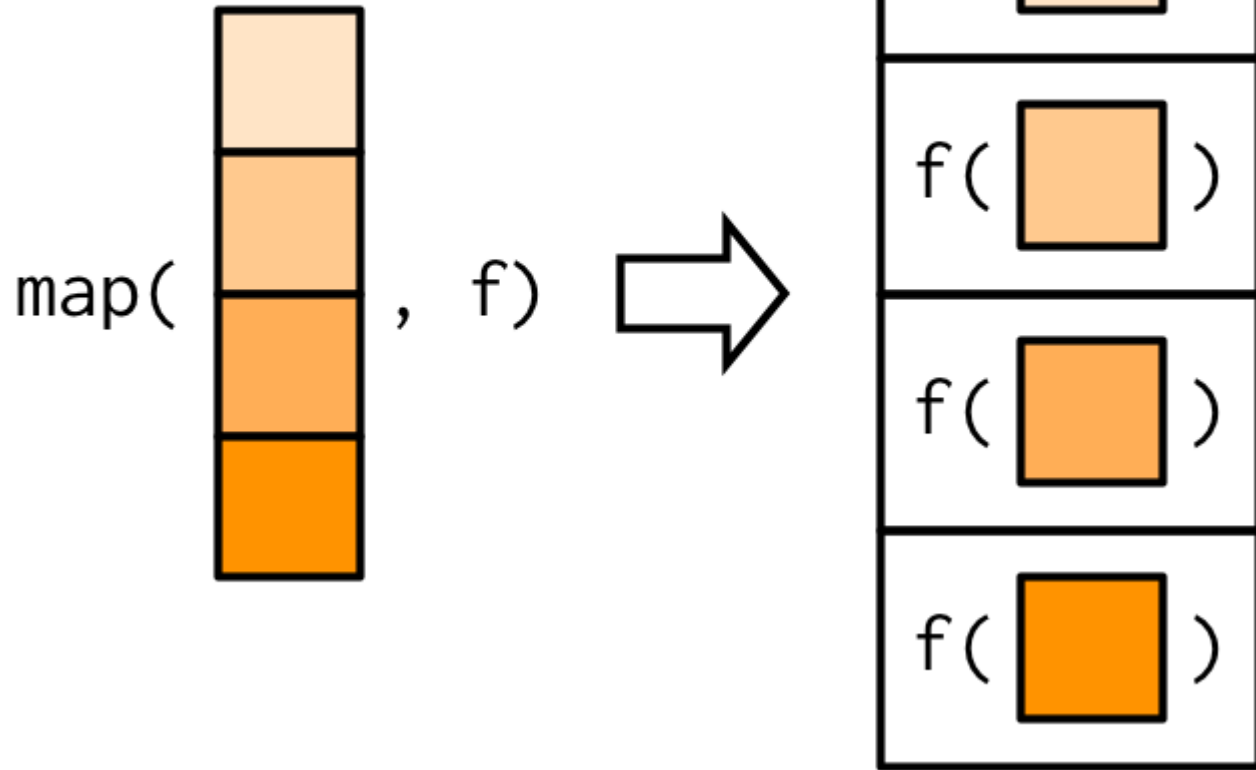
This doesn't work

```
map_dbl(smry, mtcars$mpg)
```

```
## Error: Can't pluck from a builtin
```

Why?

Remember what {purrr}
does



With `map_df`

What if we wanted this for all columns?

Challenge

- Can you extend the previous looping to supply the summary for every column? Hint: You'll need to make a nested loop (loop each function through each column)

```
map_df(mtcars, function(col) map_df(summary, ~.x(col)),
       .id = "column")
```

```
## # A tibble: 11 x 6
##   column      n n_miss n_valid      mean      sd
##   <chr>   <int>   <int>   <int>   <dbl>   <dbl>
## 1 mpg      32      0      32  20.09062  6.026948
## 2 cyl      32      0      32   6.1875   1.785922
## 3 disp     32      0      32 230.7219 123.9387
## 4 hp       32      0      32 146.6875  68.56287
## 5 drat      32      0      32   3.596562 0.5346787
## 6 wt       32      0      32   3.21725   0.9784574
## 7 qsec      32      0      32 17.84875  1.786943
## 8 vs       32      0      32   0.4375   0.5040161
## 9 am       32      0      32   0.40625   0.4989909
## 10 gear     32      0      32   3.6875   0.7378041
## 11 carb     32      0      32   2.8125   1.615200
```

05:00

Maybe easier

- Often when you get into nested loops, it's easier to turn (at least) one of the loops into a function.

```
summarize_col <- function(column) {  
  map_df(smry, ~.x(column))  
}
```

Now we can just loop this function through each column

Wrap the whole thing in a function

```
summarize_df <- function(df) {  
  map_df(df, summarize_col, .id = "column")  
}
```

```
summarize_df(airquality)
```

```
## # A tibble: 6 x 6  
##   column      n n_miss n_valid      mean      sd  
##   <chr>   <int> <int>   <int>   <dbl>   <dbl>  
## 1 Ozone    153     37    116 NA      NA  
## 2 Solar.R  153      7    146 NA      NA  
## 3 Wind     153      0    153  9.957516  3.523001  
## 4 Temp     153      0    153 77.88235  9.465270  
## 5 Month    153      0    153  6.993464  1.416522  
## 6 Day      153      0    153 15.80392  8.864520
```

Notice the missing data. Why? What should we do?


Function

components

Three components

- `body()`
- `formals()`
- `environment()` (we won't focus so much here for now)

```
poly <- function(x, power) {  
  z <- x^power  
  return(z)  
}
```



Body

Formals

Formals

- The arguments supplied to the function
- What's one way to identify the formals for a function – say, `lm`?

`?`: Help documentation!

Alternative – use a function!

```
formals(lm)
```

```
## $formula
##
##
## $data
##
##
## $subset
##
##
## $weights
```

How do you see the body?

- In RStudio: Super (command on mac, cntrl on windows) + click!

[demo]

- Alternative – just print to screen

Or use **body**

`body(lm)`

```
## {
##   ret.x <- x
##   ret.y <- y
##   cl <- match.call()
##   mf <- match.call(expand.dots = FALSE)
##   m <- match(c("formula", "data", "subset", "weights", "na.action",
##               "offset"), names(mf), 0L)
##   mf <- mf[c(1L, m)]
##   mf$drop.unused.levels <- TRUE
##   mf[[1L]] <- quote(stats::model.frame)
##   mf <- eval(mf, parent.frame())
##   if (method == "model.frame")
##     return(mf)
##   else if (method != "qr")
##     warning(gettextf("method = '%s' is not supported. Using 'qr'",
##                     method), domain = NA)
##   mt <- attr(mf, "terms")
##   y <- model.response(mf, "numeric")
##   w <- as.vector(model.weights(mf))
##   if (!is.null(w) && !is.numeric(w))
##     stop("'weights' must be a numeric vector")
##   offset <- model.offset(mf)
##   mlm <- is.matrix(y)
##   ny <- if (mlm)
```

Environment

- As I mentioned, we won't focus on this too much, but if you get deep into programming it's pretty important

```
double <- function(x) x*2  
environment(double)
```

```
## <environment: 0x7ff48c3d9f68>
```

```
environment(lm)
```

```
## <environment: namespace:stats>
```

Why this matters

What will the following return?

```
x <- 10
f1 <- function() {
  x <- 20
  x
}

f1()
```

```
## [1] 20
```

What will this return?

```
x <- 10
y <- 20
f2 <- function() {
  x <- 1
  y <- 2
  sum(x, y)
}
f2()
```

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

Last one

What do each of the following return?

```
x <- 2
f3 <- function() {
  y <- 1
  sum(x, y)
}
```

```
f3()
```

```
y
```

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

```
## Error in eval(expr, envir, enclos): object 'y' not found
```

Environment summary

- The previous examples are part of *lexical scoping*.
- Generally, you won't have to worry too much about it
- If you end up with unexpected results, this could be part of why

Scoping

- Part of what's interesting about these scoping rules is that your functions can, and very often do, depend upon things in your global workspace, or your specific environment.
- If this is the case, the function will be a "one-off", and unlikely to be useful in any other script

Example 1

Extracting information

- This is a real example

```
extract_grades <- function(dif_mod, items) {  
  item_names <- names(items)  
  delta <- -2.35*(log(dif_mod$alphaMH))  
  grades <- symnum(abs(delta),  
                   c(0, 1, 1.5, Inf),  
                   symbols = c("A", "B", "C"))  
  tibble(item = item_names, delta, grades) %>%  
    mutate(grades = as.character(grades))  
}
```


Example 2

Reading in data

```
read_sub_files <- function(file) {  
  read_csv(file) %>%  
    mutate(  
      content_area = str_extract(  
        file, "[Ee][Ll][Aa]|[Rr]dg|[Ww]ri|[Mm]ath|[Ss]ci"  
      ),  
      grade = gsub(".+g(\\d\\d*).+", "\\1", file),  
      grade = as.numeric(grade)  
    ) %>%  
    select(content_area, grade, everything()) %>%  
    clean_names()  
}  
  
ifiles <- map_df(ifiles, read_sub_files)
```

Simple example

Pull out specific coefficients

```
mods <- mtcars %>%  
  group_by(cyl) %>%  
  nest() %>%  
  mutate(  
    model = map(  
      data, ~lm(mpg ~ disp + hp + drat, data = .x)  
    )  
  )  
mods
```

```
## # A tibble: 3 x 3  
## # Groups:   cyl [3]  
##   cyl data                                model  
##   <dbl> <list>                                <list>  
## 1     6 <tibble[,10] [7 x 10]> <lm>  
## 2     4 <tibble[,10] [11 x 10]> <lm>  
## 3     8 <tibble[,10] [14 x 10]> <lm>
```

Pull a specific coef

Find the solution for one model

```
m <- mods$model[[1]]  
coef(m)
```

```
## (Intercept)          disp             hp          drat  
## 6.284507434 0.026354099 0.006229086 2.193576546
```

```
coef(m)["disp"]
```

```
##          disp  
## 0.0263541
```

```
coef(m)["(Intercept)"]
```

```
## (Intercept)  
## 6.284507
```

Generalize it

```
mods %>%  
  mutate(intercept = map_dbl(model, pull_coef, "(Intercept)"),  
         disp      = map_dbl(model, pull_coef, "disp"),  
         hp        = map_dbl(model, pull_coef, "hp"),  
         drat      = map_dbl(model, pull_coef, "drat"))
```

```
## # A tibble: 3 x 7
```

```
## # Groups:   cyl [3]
```

```
##   cyl data      model intercept      disp      hp  
##   <dbl> <list>      <list>      <dbl>      <dbl>      <dbl>  
## 1     6 <tibble[,10] [7 x 10]> <lm>      6.284507  0.02635410  0.00622908  
## 2     4 <tibble[,10] [11 x 10]> <lm>     46.08662 -0.1225361 -0.04937771  
## 3     8 <tibble[,10] [14 x 10]> <lm>     19.00162 -0.01671461 -0.02140236
```

Make it more flexible

- Since the intercept is a little difficult to pull out, we could have it return that by default.

```
pull_coef <- function(model, coef_name = "(Intercept)") {  
  coef(model)[coef_name]  
}  
mods %>%  
  mutate(intercept = map_dbl(model, pull_coef))
```

```
## # A tibble: 3 x 4  
## # Groups:   cyl [3]  
##   cyl data                model intercept  
##   <dbl> <list>                <list>      <dbl>  
## 1     6 <tibble[,10] [7 x 10]> <lm>      6.284507  
## 2     4 <tibble[,10] [11 x 10]> <lm>     46.08662  
## 3     8 <tibble[,10] [14 x 10]> <lm>     19.00162
```

Return all coefficients

```
pull_coef <- function(model) {  
  coefs <- coef(model)  
  data.frame(coefficient = names(coefs),  
             estimate     = coefs)  
}  
mods %>%  
  mutate(coefs = map(model, pull_coef))
```

```
## # A tibble: 3 x 4  
## # Groups:   cyl [3]  
##   cyl data                                model  coefs  
##   <dbl> <list>                                <list> <list>  
## 1     6 <tibble[,10] [7 x 10]> <lm>    <df[,2] [4 x 2]>  
## 2     4 <tibble[,10] [11 x 10]> <lm>    <df[,2] [4 x 2]>  
## 3     8 <tibble[,10] [14 x 10]> <lm>    <df[,2] [4 x 2]>
```

```

mods %>%
  mutate(coefs = map(model, pull_coef)) %>%
  unnest(coefs)

```

```

## # A tibble: 12 x 5
## # Groups:   cyl [3]
##   cyl data                                model coefficient      estimate
##   <dbl> <list>                                <list> <chr>          <dbl>
## 1     6 <tibble[,10] [7 x 10]> <lm> (Intercept)    6.284507
## 2     6 <tibble[,10] [7 x 10]> <lm> disp         0.02635410
## 3     6 <tibble[,10] [7 x 10]> <lm> hp           0.006229086
## 4     6 <tibble[,10] [7 x 10]> <lm> drat         2.193577
## 5     4 <tibble[,10] [11 x 10]> <lm> (Intercept) 46.08662
## 6     4 <tibble[,10] [11 x 10]> <lm> disp        -0.1225361
## 7     4 <tibble[,10] [11 x 10]> <lm> hp          -0.04937771
## 8     4 <tibble[,10] [11 x 10]> <lm> drat        -0.6041857
## 9     8 <tibble[,10] [14 x 10]> <lm> (Intercept) 19.00162
## 10    8 <tibble[,10] [14 x 10]> <lm> disp        -0.01671461
## 11    8 <tibble[,10] [14 x 10]> <lm> hp          -0.02140236
## 12    8 <tibble[,10] [14 x 10]> <lm> drat         2.006011

```

Slightly nicer

```
mods %>%  
  mutate(coefs = map(model, pull_coef)) %>%  
  select(cyl, coefs) %>%  
  unnest(coefs)
```

```
## # A tibble: 12 x 3  
## # Groups:   cyl [3]  
##   cyl coefficient      estimate  
##   <dbl> <chr>         <dbl>  
## 1     6 (Intercept)  6.284507  
## 2     6 disp         0.02635410  
## 3     6 hp          0.006229086  
## 4     6 drat        2.193577  
## 5     4 (Intercept) 46.08662  
## 6     4 disp        -0.1225361  
## 7     4 hp          -0.04937771  
## 8     4 drat        -0.6041857  
## 9     8 (Intercept) 19.00162  
## 10    8 disp        -0.01671461  
## 11    8 hp          -0.02140236  
## 12    8 drat        2.006011
```


Create nice table

```
mods %>%  
  mutate(coefs = map(model, pull_coef)) %>%  
  select(cyl, coefs) %>%  
  unnest(coefs) %>%  
  pivot_wider(names_from = "coefficient",  
              values_from = "estimate") %>%  
  arrange(cyl)
```

```
## # A tibble: 3 x 5  
## # Groups:   cyl [3]  
##   cyl `(Intercept)`      disp      hp      drat  
##   <dbl>      <dbl>      <dbl>      <dbl>      <dbl>  
## 1     4      46.08662 -0.1225361 -0.04937771 -0.6041857  
## 2     6       6.284507  0.02635410  0.006229086  2.193577  
## 3     8      19.00162 -0.01671461 -0.02140236  2.006011
```

When to write
a function?

Example

```
set.seed(42)
df <- tibble::tibble(
  a = rnorm(10, 100, 150),
  b = rnorm(10, 100, 150),
  c = rnorm(10, 100, 150),
  d = rnorm(10, 100, 150)
)
```

df

```
## # A tibble: 10 x 4
##           a           b           c           d
##       <dbl>       <dbl>       <dbl>       <dbl>
## 1  305.6438    295.7304    54.00421    168.3175
## 2   15.29527   442.9968   -167.1963    205.7256
## 3  154.4693   -108.3291    74.21240    255.2655
## 4  194.9294    58.18168   282.2012     8.661044
## 5  160.6402    80.00180   384.2790    175.7433
## 6   84.08132   195.3926    35.42963   -157.5513
## 7  326.7283    57.36206    61.40959   -17.66885
## 8   85.80114  -298.4683   -164.4745   -27.63614
## 9  402.7636   -266.0700    169.0146   -262.1311
## 10  90.59289   298.0170     4.000769   105.4184
```

Rescale each column to 0/1

Do it for one column

```
df %>%  
  mutate(a = (a - min(a, na.rm = TRUE)) /  
            (max(a, na.rm = TRUE) - min(a, na.rm = TRUE)))
```

```
## # A tibble: 10 x 4  
##       a          b          c          d  
##   <dbl>    <dbl>    <dbl>    <dbl>  
## 1 0.7493478 295.7304    54.00421 168.3175  
## 2 0         442.9968  -167.1963 205.7256  
## 3 0.3591881 -108.3291    74.21240 255.2655  
## 4 0.4636099  58.18168   282.2012    8.661044  
## 5 0.3751145  80.00180   384.2790   175.7433  
## 6 0.1775269 195.3926    35.42963 -157.5513  
## 7 0.8037639  57.36206    61.40959 -17.66885  
## 8 0.1819655 -298.4683  -164.4745  -27.63614  
## 9 1        -266.0700   169.0146  -262.1311  
## 10 0.1943323 298.0170     4.000769 105.4184
```

Do it for all columns

```
df %>%  
  mutate(a = (a - min(a, na.rm = TRUE)) /  
            (max(a, na.rm = TRUE) - min(a, na.rm = TRUE)),  
         b = (b - min(b, na.rm = TRUE)) /  
            (max(b, na.rm = TRUE) - min(b, na.rm = TRUE)),  
         c = (c - min(c, na.rm = TRUE)) /  
            (max(c, na.rm = TRUE) - min(c, na.rm = TRUE)),  
         d = (d - min(d, na.rm = TRUE)) /  
            (max(d, na.rm = TRUE) - min(d, na.rm = TRUE)))
```

```
## # A tibble: 10 x 4  
##       a          b          c          d  
##   <dbl>    <dbl>    <dbl>    <dbl>  
## 1 0.7493478 0.8013846 0.4011068 0.8319510  
## 2 0          1          0          0.9042516  
## 3 0.3591881 0.2564372 0.4377506 1  
## 4 0.4636099 0.4810071 0.8149005 0.5233744  
## 5 0.3751145 0.5104355 1          0.8463031  
## 6 0.1775269 0.6660608 0.3674252 0.2021270  
## 7 0.8037639 0.4799017 0.4145351 0.4724852  
## 8 0.1819655 0          0.004935493 0.4532209  
## 9 1          0.04369494 0.6096572 0  
## 10 0.1943323 0.8044685 0.3104346 0.7103825
```

An alternative

- What's an alternative we could use *without* writing a function?

```
map_df(df, ~(.x - min(.x, na.rm = TRUE)) /  
          (max(.x, na.rm = TRUE) - min(.x, na.rm = TRUE)))
```

```
## # A tibble: 10 x 4  
##       a           b           c           d  
##   <dbl>   <dbl>   <dbl>   <dbl>  
## 1 0.7493478 0.8013846 0.4011068 0.8319510  
## 2 0         1         0         0.9042516  
## 3 0.3591881 0.2564372 0.4377506 1  
## 4 0.4636099 0.4810071 0.8149005 0.5233744  
## 5 0.3751145 0.5104355 1         0.8463031  
## 6 0.1775269 0.6660608 0.3674252 0.2021270  
## 7 0.8037639 0.4799017 0.4145351 0.4724852  
## 8 0.1819655 0         0.004935493 0.4532209  
## 9 1         0.04369494 0.6096572 0  
## 10 0.1943323 0.8044685 0.3104346 0.7103825
```

Another alternative

Write a function

- What are the arguments going to be?
- What will the body be?

Arguments

- One formal argument – A numeric vector to rescale

Body

- You try first

```
(x - min(x, na.rm = TRUE)) /  
  (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
```

02:00

Create the function

Test it!

```
rescale01(c(0, 5, 10))
```

```
## [1] 0.0 0.5 1.0
```

```
rescale01(c(seq(0, 100, 10)))
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Make it cleaner

- There's nothing inherently "wrong" about the prior function, but it is a bit hard to read
- How could we make it easier to read?
 - Remove missing data once (rather than every time)
 - Don't calculate things multiple times

A little cleaned up

```
rescale01b <- function(x) {  
  z <- na.omit(x)  
  min_z <- min(z)  
  max_z <- max(z)  
  
  (z - min_z) / (max_z - min_z)  
}
```

Test it!

```
rescale01b(c(0, 5, 10))
```

```
## [1] 0.0 0.5 1.0
```

```
rescale01b(c(seq(0, 100, 10)))
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Make sure they give the same output

```
identical(rescale01(c(0, 1e5, .01)), rescale01b(c(0, 1e5, 0.01)))
```

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

```
rand <- rnorm(1e3)  
identical(rescale01(rand), rescale01b(rand))
```

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

Final solution

Could use **modify** here too

```
map_df(df, rescale01b)
```

```
## # A tibble: 10 x 4
##       a          b          c          d
##   <dbl>    <dbl>    <dbl>    <dbl>
## 1 0.7493478 0.8013846 0.4011068 0.8319510
## 2 0          1          0          0.9042516
## 3 0.3591881 0.2564372 0.4377506 1
## 4 0.4636099 0.4810071 0.8149005 0.5233744
## 5 0.3751145 0.5104355 1          0.8463031
## 6 0.1775269 0.6660608 0.3674252 0.2021270
## 7 0.8037639 0.4799017 0.4145351 0.4724852
## 8 0.1819655 0          0.004935493 0.4532209
## 9 1          0.04369494 0.6096572 0
## 10 0.1943323 0.8044685 0.3104346 0.7103825
```

Getting more complex

- What if you want a function to behave differently depending on the input?

Add conditions

```
function() {  
  if (condition) {  
  
    # code executed when condition is TRUE  
  
  } else {  
    # code executed when condition is FALSE  
  
  }  
}
```

Lots of conditions?

```
function() {  
  if (this) {  
  
    # do this  
  
  } else if (that) {  
  
    # do that  
  
  } else {  
  
    # something else  
  
  }  
}
```

Easy example

- Given a vector, return the mean if it's numeric, and **NULL** otherwise

```
mean2 <- function(x) {  
  if(is.numeric(x)) {  
    mean(x)  
  }  
  else {  
    return()  
  }  
}
```


Test it

```
mean2(rnorm(12))
```

```
## [1] 0.1855869
```

```
mean2(letters[1:5])
```

```
## NULL
```

Mean for all numeric columns

- The prior function can now be used within a new function to calculate the mean of all columns of a data frame that are numeric

```
means_df <- function(df) {  
  means <- map(df, mean2) # calculate means  
  nulls <- map_lgl(means, is.null) # find null values  
  means_l <- means[!nulls] # subset list to remove nulls  
  
  as.data.frame(means_l) # return a df  
}
```

```
head(iris)
```

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

```
means_df(iris)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      5.843333     3.057333     3.758     1.199333
```

We have a problem though!

```
head(airquality)
```

```
##      Ozone  Solar.R Wind  Temp Month  Day
## 1      41      190  7.4    67     5    1
## 2      36      118  8.0    72     5    2
## 3      12      149 12.6    74     5    3
## 4      18      313 11.5    62     5    4
## 5      NA       NA 14.3    56     5    5
## 6      28       NA 14.9    66     5    6
```

```
means_df(airquality)
```

```
##      Ozone  Solar.R      Wind      Temp      Month      Day
## 1      NA       NA 9.957516 77.88235 6.993464 15.80392
```

Why is this happening?

How can we fix it?

Easiest way in this case . . .

Pass the dots!

Redefine **means2**

```
mean2 <- function(x, ...) {  
  if(is.numeric(x)) {  
    mean(x, ...)  
  }  
  else {  
    return()  
  }  
}
```

Reefine means_df

```
means_df <- function(df, ...) {  
  means <- map(df, mean2, ...) # calculate means  
  nulls <- map_lgl(means, is.null) # find null values  
  means_l <- means[!nulls] # subset list to remove nulls  
  
  as.data.frame(means_l) # return a df  
}
```

```
means_df(airquality)
```

```
##      Ozone Solar.R      Wind      Temp      Month      Day  
## 1      NA      NA 9.957516 77.88235 6.993464 15.80392
```

```
means_df(airquality, na.rm = TRUE)
```

```
##      Ozone Solar.R      Wind      Temp      Month      Day  
## 1 42.12931 185.9315 9.957516 77.88235 6.993464 15.80392
```

Next time

Functions: Part 2