Returning values from functions

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

datacamp

A simple sum function

```
simple_sum <- function(x) {
  if(anyNA(x)) {
    return(NA)
  }
  total <- 0
  for(value in x) {
    total <- total + value
  }
  total
}
```

```
simple_sum(c(0, 1, 3, 6, NA, 7))
```

```
NA
```

Geometrics means again

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  assert_is_numeric(x)
  if(any(is_non_positive(x), na.rm = TRUE)) {
    stop("x contains non-positive values, so the geometric mean makes no sense.")
  }
  na.rm <- coerce_to(use_first(na.rm), "logical")
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

Returning NaN with a warning

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  assert_is_numeric(x)
  if(any(is_non_positive(x), na.rm = TRUE)) {
    warning("x contains non-positive values, so the geometric mean makes no sense.")
    return(NaN)
  }
  na.rm <- coerce_to(use_first(na.rm), "logical")
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

Reasons for returning early

1. You already know the answer.
2. The input is an edge case.

Hiding the return value

```
simple_sum <- function(x) {
  if(anyNA(x)) {
    return(NA)
  }
  total <- 0
  for(value in x) {
    total <- total + value
  }
  total
}
```

```
simple_sum(c(0, 1, 3, 6, 2, 7))
```

```
19
```

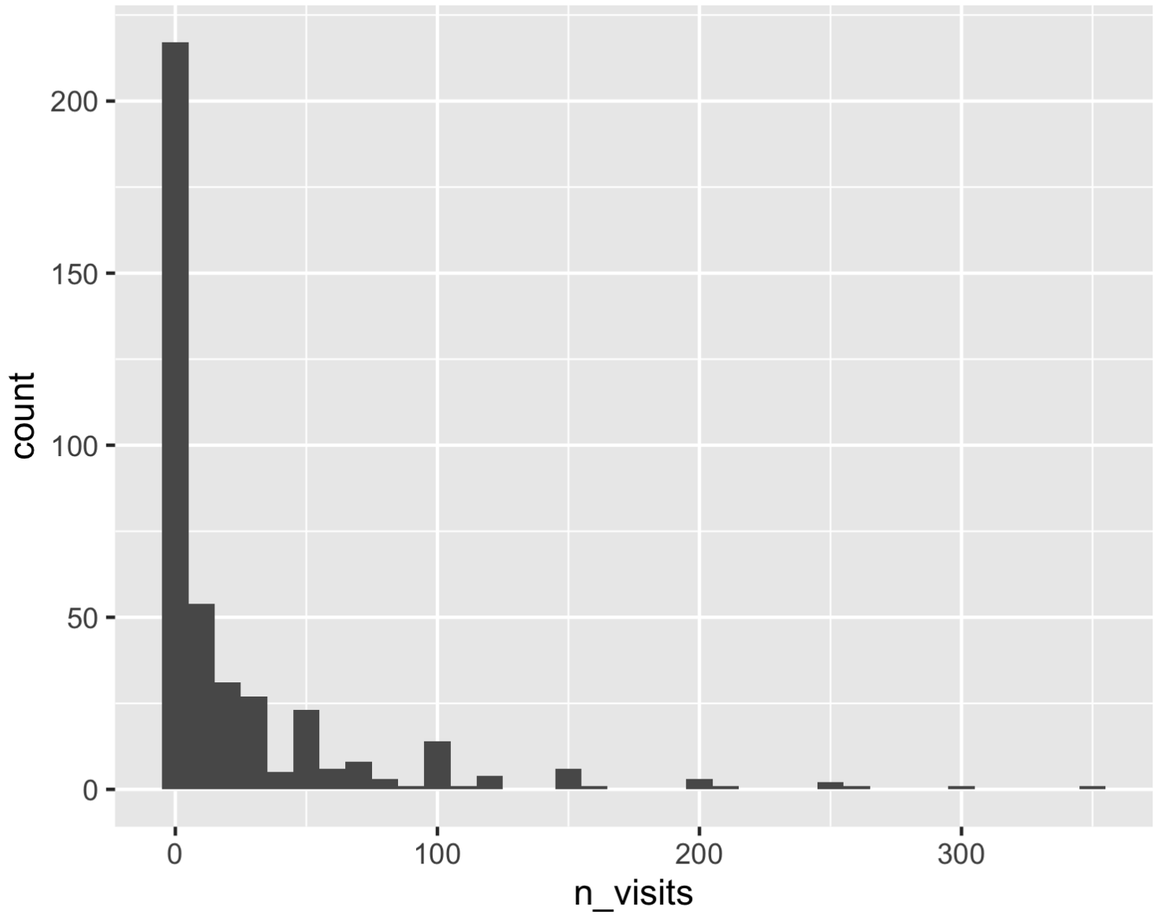
Hiding the return value

```
simple_sum <- function(x) {
  if(anyNA(x)) {
    return(NA)
  }
  total <- 0
  for(value in x) {
    total <- total + value
  }
  invisible(total)
}
```

```
simple_sum(c(0, 1, 3, 6, 2, 7))
```

Many plots invisibly return things

```
ggplot(snake_river_visits, aes(n_visits)) +
  geom_histogram(binwidth = 10)
```

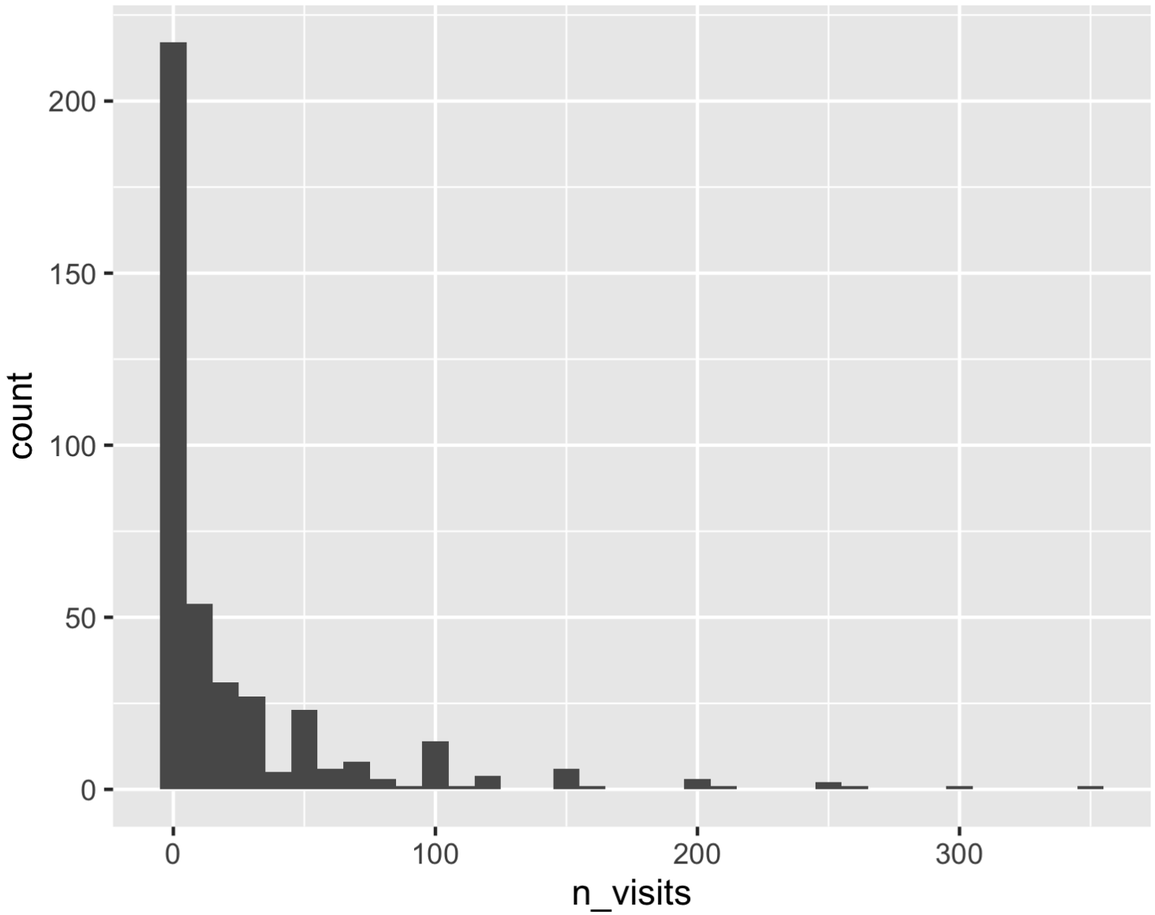


Many plots invisibly return things

```
srv_hist <- ggplot(snake_river_visits, aes(n_visits)) +
  geom_histogram(binwidth = 10)
```

```
str(srv_hist, max.level = 0)
```

```
List of 9
 - attr(*, "class")= chr [1:2] "gg" "ggplot"
```



Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R

Returning multiple values from functions

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

datacamp

Getting the session information

```
R.version.string
```

```
"R version 3.5.3 (2019-03-11)"
```

```
Sys.info()[c("sysname", "release")]
```

```
sysname                          release
"Linux" "4.14.106-79.86.amzn1.x86_64"
```

```
loadedNamespaces()
```

```
 [1] "Rcpp"       "grDevices"  "crayon"
 [4] "dplyr"      "assertthat" "R6"
 [7] "magrittr"   "datasets"   "pillar"
[10] "rlang"      "utils"      "praise"
[13] "rstudioapi" "graphics"   "base"
[16] "tools"      "glue"       "purrr"
[19] "yaml"       "compiler"   "pkgconfig"
[22] "stats"      "tidyselect" "methods"
[25] "tibble"
```

Defining session()

```
session <- function() {
  r_version <- R.version.string,
  operating_system <- Sys.info()[c("sysname", "release")],
  loaded_pkgs <- loadedNamespaces()
  # ???
}
```

Defining session()

```
session <- function() {
  list(
    r_version = R.version.string,
    operating_system = Sys.info()[c("sysname", "release")],
    loaded_pkgs = loadedNamespaces()
  )
}
```

Calling session()

```
session()
```

```
$r_version
[1] "R version 3.5.3 (2019-03-11)"

$operating_system
sysname                       release
"Linux" "4.14.106-79.86.amzn1.x86_64"

$loaded_pkgs
 [1] "Rcpp"       "grDevices"  "crayon"
 [4] "dplyr"      "assertthat" "R6"
 [7] "magrittr"   "datasets"   "pillar"
[10] "rlang"      "utils"      "praise"
[13] "rstudioapi" "graphics"   "base"
[16] "tools"      "glue"       "purrr"
[19] "yaml"       "compiler"   "pkgconfig"
[22] "stats"      "tidyselect" "methods"
[25] "tibble"
```

Multi-assignment

```
library(zeallot)
c(vrsn, os, pkgs) %<-% session()
```

```
vrsn
```

```
"R version 3.5.3 (2019-03-11)"
```

```
os
```

```
sysname                        release
"Linux" "4.14.106-79.86.amzn1.x86_64"
```

Attributes

```
month_no <- setNames(1:12, month.abb)
month_no
```

```
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
  1   2   3   4   5   6   7   8   9  10  11  12
```

```
attributes(month_no)
```

```
$names
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul"
 [8] "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
attr(month_no, "names")
```

```
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul"
 [8] "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
attr(month_no, "names") <- month.name
month_no
```

```
  January  February     March     April       May
        1         2         3         4         5
     June      July    August September   October
        6         7         8         9        10
 November  December
       11        12
```

Attributes of a data frame

```
orange_trees
```

```
# A tibble: 35 x 3
   Tree    age circumference
   <ord> <dbl>         <dbl>
 1 1       118            30
 2 1       484            58
 3 1       664            87
 4 1      1004           115
 5 1      1231           120
 6 1      1372           142
 7 1      1582           145
 8 2       118            33
 9 2       484            69
10 2       664           111
# … with 25 more rows
```

```
attributes(orange_trees)
```

```
$names
[1] "Tree"          "age"
[3] "circumference"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[16] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
[31] 31 32 33 34 35

$class
[1] "tbl_df"     "tbl"        "data.frame"
```

[1] data(Orange, package = "datasets")

Attributes added by group_by()

```
library(dplyr)
orange_trees %>%
  group_by(Tree) %>%
  attributes()
```

```
$names
[1] "Tree"          "age"           "circumference"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

$class
[1] "grouped_df" "tbl_df"     "tbl"        "data.frame"

$groups
# A tibble: 5 x 2
  Tree  .rows
  <ord> <list>
1 3     <int [7]>
2 1     <int [7]>
3 5     <int [7]>
4 2     <int [7]>
5 4     <int [7]>
```

When to use each technique

- If you need the result to have a particular type, add additional return values as attributes.
- Otherwise, collect all return values into a list.

broom

Model objects are converted into 3 data frames.

function	level	example
`glance()`	model	degrees of freedom
`tidy()`	coefficient	p-values
`augment()`	observation	residuals

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R

Environments

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

Environments are like lists

```
datacamp_lst <- list(
  name = "DataCamp",
  founding_year = 2013,
  website = "https://www.datacamp.com"
)
```

```
datacamp_env <- list2env(datacamp_lst)
```

```
ls.str(datacamp_lst)
```

```
ls.str(datacamp_env)
```

```
founding_year :  num 2013
name :  chr "DataCamp"
website :  chr "https://www.datacamp.com"
```

```
founding_year :  num 2013
name :  chr "DataCamp"
website :  chr "https://www.datacamp.com"
```

Environments have parents

Getting the parent environment

```
parent <- parent.env(datacamp_env)
environmentName(parent)
```

```
"R_GlobalEnv"
```

```
grandparent <- parent.env(parent)
environmentName(grandparent)
```

```
"package:stats"
```

```
search()
```

```
[1] ".GlobalEnv"        "package:stats"
[3] "package:graphics"  "package:grDevices"
[5] "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"
[9] "package:base"
```

Does a variable exist?

```
datacamp_lst <- list(
  name = "DataCamp",
  website = "https://www.datacamp.com"
)
datacamp_env <- list2env(datacamp_lst)
founding_year <- 2013
```

```
exists("founding_year", envir = datacamp_env)
```

```
TRUE
```

```
exists("founding_year", envir = datacamp_env, inherits = FALSE)
```

```
FALSE
```

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R

Scope and precedence

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

Accessing variables outside functions

```
x_times_y <- function(x) {
  x * y
}
```

```
x_times_y(10)
```

```
Error in x_times_y(10) :
  object 'y' not found
```

```
x_times_y <- function(x) {
  x * y
}
y <- 4
```

```
x_times_y(10)
```

```
40
```

Accessing function variables from outside

```
x_times_y <- function(x) {
  x * y
}
y <- 4
x_times_y(10)
```

```
print(x)
```

```
Error in print(x) : object 'x' not found
```

What's best? Inside or outside?

```
x_times_y <- function(x) {
  y <- 6
  x * y
}
y <- 4
```

```
x_times_y(10)
```

```
60
```

Passed in vs. defined in

```
x_times_y <- function(x) {
  x <- 9
  y <- 6
  x * y
}
y <- 4
```

```
x_times_y(10)
```

```
54
```

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R