Announcements

1. Will circulate final project guidelines next week
2. No homework for next class (2/17), but there will be one for the following week (2/24) on writing functions

Today

1. Workflow self-critique presentations
2. Break
3. Introductions to functions + tutorial

PSYC 259: Principles of Data Science

Week 6: Writing custom functions

What are functions?

Functions

- A series of code statements designed to accomplish a particular task
- Functions take input(s) and return output(s)
- mean(x, na.RM = FALSE)
  - fx name = mean
  - fx arguments = x, na.RM
  - fx body = code used to calculate the mean
  - fx output = the mean value that is returned

Defining your “hyp” function

a name you assign

```
hyp <- function(side_a, side_b) {
  a <- side_a^2
  b <- side_b^2
  h <- sqrt(a + b)
  return(h)
}
```

Defining your “hyp” function

a name you assign

argument names you assign

```
hyp <- function(side_a, side_b) {
  a <- side_a^2
  b <- side_b^2
  h <- sqrt(a + b)
  return(h)
}
```

Defining your “hyp” function

a name you assign

argument names you assign

code body

```
hyp <- function(side_a, side_b) {
  a <- side_a^2
  b <- side_b^2
  h <- sqrt(a + b)
  return(h)
}
```

Defining your "hyp" function

a name you assign

argument names you assign

code body

```
hyp <- function(side_a, side_b) {
  a <- side_a^2
  b <- side_b^2
  h <- sqrt(a + b)
  return(h)
}
```

define the output value

Defining your "hyp" function

a name you assign

argument names you assign

code body

```
hyp <- function(side_a, side_b) {
  a <- side_a^2
  b <- side_b^2
  h <- sqrt(a + b)
  return(h)
}
```

define the output value

```
> hyp(3,4)
[1] 5
```

use like any other function

Why do we write functions?

- Save time/typing
  - We could all write code to calculate a mean, but imagine doing that every time we needed to

```
> x <- c(1, 2, 3, 4, 5)
> mean_x <- sum(x)/length(x)
> y <- c(3, 4, 5, 6, 7)
> mean_y <- sum(y)/length(y)
> z <- c(10, 11, 12, 13, 14, 15)
> mean_z <- sum(z)/length(z)
> mean_x
[1] 3
> mean_y
[1] 5
> mean_z
[1] 12.5
```

```
> vector_mean <- function(v) sum(v)/length(v)
> vector_mean(x)
[1] 3
> vector_mean(y)
[1] 5
> vector_mean(z)
[1] 12.5
```

Why do we write functions?

- Save time/typing
  - We could all write code to calculate a mean, but imagine doing that every time we needed to
- Encapsulate computation
  - Functions execute in their own *environment*
  - We just want the output, not all of the intermediary steps of the computation
- Introduce abstraction
  - Good functions are general-purpose tools

Packages extend R functionality with functions

- base package = functions for data types, basic math, and other generic functions
- other packages = sets of functions that extend the base R language
  - May be written using base R functions or a mix of base R and functions from other packages
  - dplyr depends on base R but also on other packages (such as tibble and tidyselect)

Packages are just a set of function definitions

- calling “arrange” before library(dplyr) won’t work, because arrange is defined by dplyr
- calling “vector_mean” before assigning a function to vector_mean won’t work for the same reason
- Loading a package with library() is just a shortcut for defining lots of functions

What’s special about packages?

- The short answer: nothing
  - The functions you write vs. use from a package operate the same way when you execute them
  - Just because a function is in a package does not guarantee that it “works"

What's special about packages?

- The longer answer
  - Packages on CRAN can be installed with install.packages
  - Packages on CRAN have to pass a stringent set of automated checks every day to make sure they run
  - Large user bases make it more likely that errors are found
  - devtools::install_github() lets you install not-vetted packages, which may be experimental/buggy

Other thoughts

- There's little cost to writing functions
  - Even if it's something you'll use 3-4 times, it could clean up your code and make it easier to run
- Write functions that do one thing well
  - Functions that are trying to do too many things can be hard to name and hard to debug
- Revise your functions over time
  - Add arguments to make them more general