# Lecture 17 – Debugging

## Learning Objectives:

1. Become proficient in the use of the R language

   1.8 Learn the basics of debugging interactively and non-interactively in R.

3. Learn the basic principles of software design.

   3.6 Learn effective strategies for debugging code interactively and non-interactively.
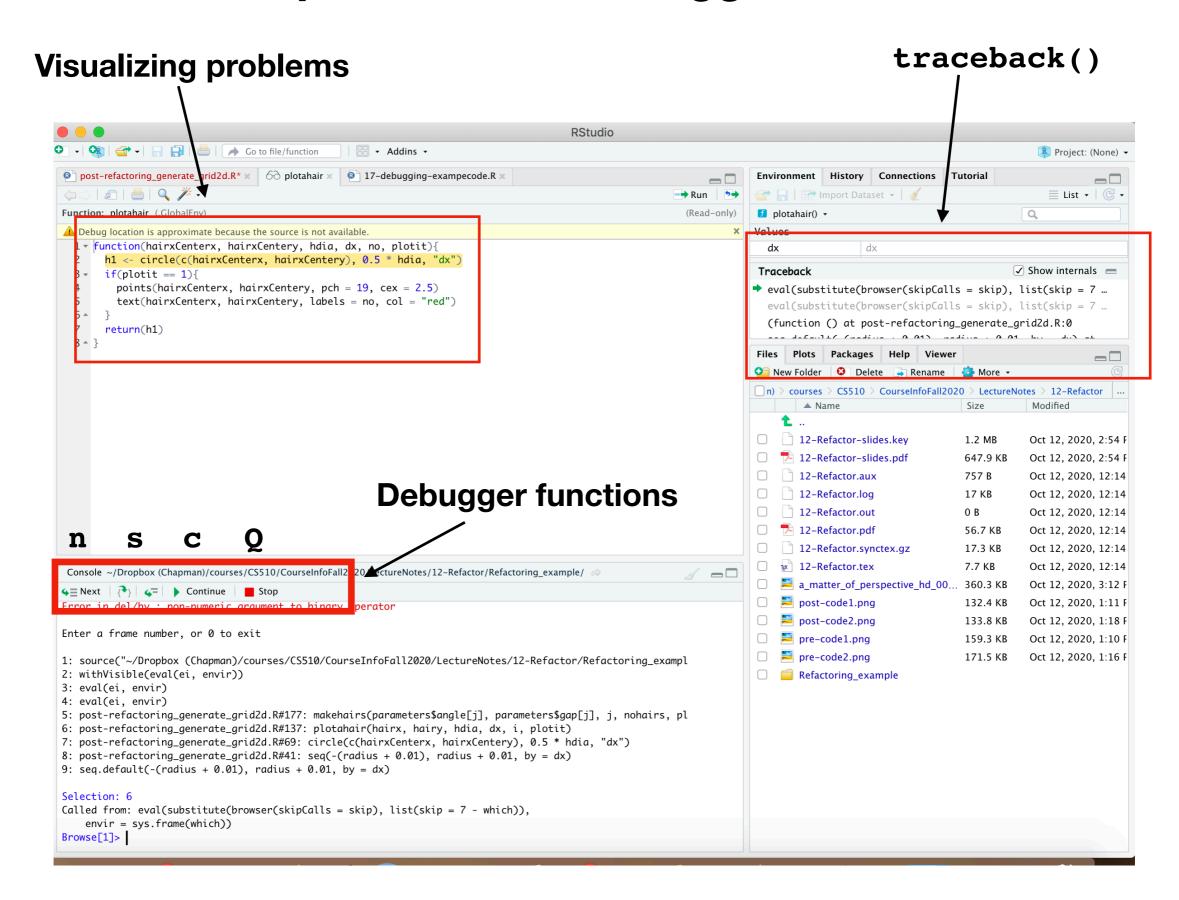
   3.7 Practice defensive coding.

## Midterm survey is on Canvas!

# Debugging strategy

**Hadley Wickham's Steps for Debugging (from *Advanced R*)**

1. **Google it!** Googling will sometimes yield a solution.

    - Remove variable and function names
    - Read and understand the post, why the error occurs, and why the fix works.
    - Read a variety of posts to see if the fixes are consistent or if there are other options.

2. **Make it repeatable.** Form a hypothesis and test it until it works out.

    - Copy code into separate script and run it that way.
    - Create an automated test.

3. **Find out where it is.** Critical step to find where in the script the error originates.

    - Use traceback() to find error
    - For functions, define parameters manually and run line-by-line
    - For for loops, define your looped parameter and run line-by-line
    - There are tools!

4. **Fix it and test it.**  Be sure to make the error a test!

# Tools that will help: RStudio's debugger

# Non-interactive debugging

Stick with the general strategy, but there are fewer tools to help.

Check common problems:

- Did you run the code in a clean session?
- Is the working directory different?
- Is the path environment different?
- Is the `R_LIBS` path different?

Use `dump.frames()` (like `recover()` in interactive mode):

```r
dump_and_quit <- function() {
    # Save debugging info to file las.dump.rda
    dump.frames(to.file = TRUE)
    # Quits R with error status
    q(status = 1)
}

options(error = dump_and_quit)
```
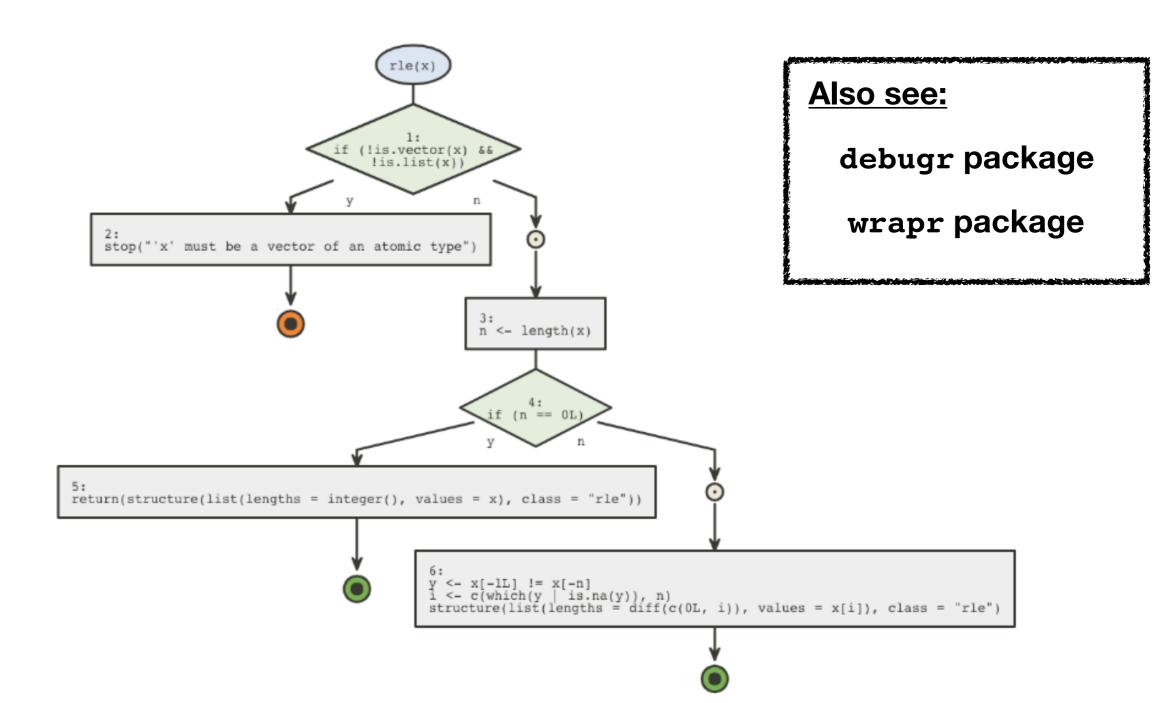
# Tools that will help: `flow` package

**To install:** `remotes::install_github("moodymudskipper/flow")`

Draws flow charts of functions. Useful for a deep dive into analysis pipelines.

**https://moodymudskipper.github.io/flow/articles/Draw-a-function.html**

rle(x)

1:
if (!is.vector(x) &&
!is.list(x))

y          n

2:
stop("'x' must be a vector of an atomic type")

3:
n <- length(x)

4:
if (n == 0L)

y          n

5:
return(structure(list(lengths = integer(), values = x), class = "rle"))

6:
y <- x[-1L] != x[-n]
i <- c(which(y | is.na(y)), n)
structure(list(lengths = diff(c(0L, i)), values = x[i]), class = "rle")

**Also see:**

**debugr package**

**wrapr package**

# Defensive programming

**"Fail fast."**

- Be strict about what you accept.

- Avoid functions that use non-standard evaluation.

- Avoid functions that return different types of output depending on their input.

**Condition handling:**

- `try()`

- `tryCatch()`

# Additional Resources

https://adv-r.hadley.nz/debugging.html#non-interactive-debugging –
Debugging, *Advanced R*

https://data-flair.training/blogs/debugging-in-r-programming/ –
R Debug Essentials

https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio –
Debugging with RStudio

https://rdrr.io/cran/wrapr/f/vignettes/DebugFnW.Rmd – wrapr Debug Vignette

https://cran.r-project.org/web/packages/debugr/vignettes/debugr.html –
debugr package vignette

# Code Peer-Review Form:

**https://forms.gle/F22aWV8VKGjqvoaa6**

**Procedure for peer-reviewing code:**

**1. Download the release specified on Canvas.**

**2. Open and enter the directory.**

**3. Read and follow the instructions in the README.**

**4. If no instructions are provided, either:**
   **• open RStudio project and source first R script.**
   **• setwd() and source first R script.**

**5. Fill out the form.**