

# Stat405

Debugging

Hadley Wickham

# Pop quiz

Which is the best way to solve a problem?

Write a giant function that tries to do everything, and then when it doesn't work you have no idea where the problem is.

Write small functions that each do a single task and can be tested easily. If there is a problem, you can localise it to a few lines of code

**Tools of  
last resort**

If you're using  
them all the time,  
something is  
wrong with your  
basic approach

`traceback()`

`browser()`

`recover()`

`options(error)`

```
f <- function(x) {  
  h(x)  
  
  i(x)  
}  
g <- function(x) {  
  a <- 10  
  x  
}  
h <- function(x) {  
  i(x)  
  i(x)  
}  
i <- function(x) {  
  if (sample(5, 1) == 1) stop("This is an error!")  
}  
  
f()  
traceback() # This is the call stack
```

# Browser

Creates an interactive prompt in a function's environment

Four special commands (no brackets):

n = next line

c = continue (or just press return)

where = where am I in the call stack?

Q = quit

Can also use any regular R function

```
j <- function(x, y = 10) {  
  k(x, y)  
}
```

```
k <- function(x, y) {  
  z <- 3  
  browser()  
  x + y  
}
```

```
j(10)
```

# Your turn

Familiarise yourself with `browser()`

Try using `ls()` while you are browsing.  
What do you see?

Try modifying the values inside the  
function. What happens to the result?

# Browser

`debug(f)` automatically adds a browser statement to the start of `f`, `undebbug(f)` removes it.

`debugonce(f)` automatically removes it after `f` is run for the first time

If the error occurs only under some conditions you might want to put `browser()` inside an `if` statement.

# Recover

Works like `browser()`, but lets you jump anywhere in the call stack

Most usefully:

```
options(error = recover)
```

```
# Can change the default behaviour when an error occurs
```

```
options(error = recover)
```

```
f()
```

```
# Set to NULL to return to the default (i.e. do nothing)
```

```
options(error = NULL)
```

```
# Another useful option: turn warnings into errors
```

```
options(warn = 2)
```

```
# and turn them back
```

```
options(warn = 0)
```

# Your turn

Use `options(error = recover)` and explore the call stack of `f`.

Use `ls()` to explore what variables are defined in each environment

```
# Your turn
```

```
# Use the tools you have just learned about to debug
```

```
# this function and create a version that works
```

```
larger <- function(x, y) {  
  y.is.bigger <- y > x  
  x[y.is.bigger] <- y[y.is.bigger]  
  x  
}  
larger(c(1, 5, 10), c(2, 4, 11))  
larger(c(1, 5, 10), 6)  
larger(c(1, 5, 10), c(2, 4, NA))
```

# Learn more

<http://www.biostat.jhsph.edu/~rpeng/docs/R-debug-tools.pdf>