Debugging with browser

## Learning Objectives

* Use browser to set breakpoints
* Use browser to set conditional breakpoints

In the last lesson we used debug to enter into a function's environment for interactive debugging. However, if we have an idea where the bug is located, we can use the function browser to set a "breakpoint" in that location. This prevents us from having to step through each line of a function to reach the point where the problem is located. Furthermore, we can use a conditional statement to only activate the debugger when a certain condition is true (especially useful for long for loops).

## Turn off RStudio's debugging features

Just like last lesson, make sure to turn of RStudio's debugging features. In the menu, go to "Debug". From the dropdown menu, go to "On Error" and choose the setting "Message Only".

We'll start with the function we updated in the last lesson.

mean\_metric\_per\_var <- function(metric, variable) {  
 if (!is.factor(variable)) {  
 variable <- as.factor(variable)  
 }  
 variable <- droplevels(variable)  
 result <- numeric(length = length(levels(variable)))  
 names(result) <- levels(variable)  
 for (v in levels(variable)) {  
 result[v] <- mean(metric[variable == v])  
 }  
 return(result)  
}

And we'll focus on fixing the following behavior:

mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

pbio pcbi pgen pmed pntd pone ppat   
0.2041683 0.1906311 0.2006783 0.2293156 0.2297650 NA 0.1893234

Our function returns NA for the total number of Facebook likes for PLOS One. Why is this happening? It is correctly identifying all the journals, so the first section of code appears to be working correctly. The totals per journal are computed within the for loop, so that is likely where the problem is originating. Since we suspect the problem is occuring during the for loop, we'll set the breakpoint there with browser instead of starting from the beginning of the function using debug.

mean\_metric\_per\_var <- function(metric, variable) {  
 if (!is.factor(variable)) {  
 variable <- as.factor(variable)  
 }  
 variable <- droplevels(variable)  
 result <- numeric(length = length(levels(variable)))  
 names(result) <- levels(variable)  
 for (v in levels(variable)) {  
 browser()  
 result[v] <- mean(metric[variable == v])  
 }  
 return(result)  
}

Now the next time we call the function, we are dropped into the debugger at the breakpoint set by browser.

mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Called from: mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Browse[1]>

Let's confirm that the beginning of the function has already been run.

Browse[1]> ls()

[1] "metric" "result" "v" "variable"

Furthermore, we can check the current value of v in the for loop.

Browse[1]> v

[1] "pbio"

## Investigating variables with the same names as debugging commands

What if you use one of the debugging commands, e.g. n, s, f, as one of the names of your variables? If you type them into the debugger, they will be used as commands. If you want to view the value contained by the variable in the debugger, use print, e.g. print(n).

We want to see what is happening when v is "pone". As we did before we could step through line by line using n. But using this approach, each time through the loop we would have to type n multiple times to run the lines of code in the loop. This would be even worse if there were many lines of code inside. Instead, we can use c for "continue", which continues running the code until the next time browser is called.

Browse[1]> c

Called from: mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Browse[1]> v

[1] "pcbi"

Browse[1]> c

Called from: mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Browse[2]> v

[1] "pgen"

But this really isn't much better, especially if we run through the for loop multiple times as we attempt to debug the function. Let's quit the debugger and try a new strategy.

Browse[2]> Q

We can set a conditional breakpoint using an if statement. Then we will be dropped into the interactive debugger only when the condition is true. We want to enter the debugger when v == "pone".

mean\_metric\_per\_var <- function(metric, variable) {  
 if (!is.factor(variable)) {  
 variable <- as.factor(variable)  
 }  
 variable <- droplevels(variable)  
 result <- numeric(length = length(levels(variable)))  
 names(result) <- levels(variable)  
 for (v in levels(variable)) {  
 if (v == "pone") {  
 browser()  
 }  
 result[v] <- mean(metric[variable == v])  
 }  
 return(result)  
}

mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Called from: mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

Browse[1]> v

[1] "pone"

Now we entered into the debugger after the loop has reached "pone". Let's inspect the variable the values being passed to mean. Specifically, let's see all the unique values.

Browse[1]> unique(metric[variable == v])

[1] 0 2 1 6 5 3 10 7 4 8 35 12 34 9 37 16 14 25 892 22  
[21] 11 19 18 13 21 49 15 41 50 17 51 104 23 20 26 151 30 39 NA 24  
[41] 95 44 109 66

Interestingly, at least one of the Facebook Like counts are NA. Is this different from the other journals? Let's check "pbio".

Browse[1]> unique(metric[variable == "pbio"])

[1] 0 1 2 3 4 12 8 39 7 5 10 45 11 18 44 6 14 13 43 19

It does not contain any NAs, so this is likely the problem.

Let's first exit the debugging environment.

Browse[1]> Q

And then check the help for mean to see if we can figure out what is going on (remember you can also press the F1 key to see a function's help page).

?mean

From the help page, we see that mean has an argument na.rm to remove NAs.

na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.

Otherwise by default, the function returns NA if any of the values are NA.

Let's update the function so that mean removes NA values. At this point we can also remove the call to browser.

mean\_metric\_per\_var <- function(metric, variable) {  
 if (!is.factor(variable)) {  
 variable <- as.factor(variable)  
 }  
 variable <- droplevels(variable)  
 result <- numeric(length = length(levels(variable)))  
 names(result) <- levels(variable)  
 for (v in levels(variable)) {  
 result[v] <- mean(metric[variable == v], na.rm = TRUE)  
 }  
 return(result)  
}

And now the function works properly when passed an NA!

mean\_metric\_per\_var(counts\_raw$facebookLikeCount, counts\_raw$journal)

pbio pcbi pgen pmed pntd pone ppat   
0.2041683 0.1906311 0.2006783 0.2293156 0.2297650 0.3519648 0.1893234