

Week 5 Tutorial: Debugging and Testing in R

POP77001 Computer Programming for Social Scientists

Debugging with `print()`

- `print()` statement can be used to check the internal state of a program during evaluation.
- Can be placed in critical parts of code (before or after loops/function calls/objects loading).
- For harder cases switch to R debugger.

Exercise: Debugging

- See the function for calculating Pearson correlation below.
- Recall that sample correlation can be calculated using this formula:

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the means (averages) of variable x and y , respectively.

- What do you think is the correlation coefficient between vectors `c(1, 2, 3, 4, 5)` and `c(-3, -5, -7, -9, -11)`?
- Check the output of the function, is it correct?
- Find and fix any problems that you encounter

```

1 pearson <- function(vec_x, vec_y) {
2   if (!(is.numeric(vec_x) && is.numeric(vec_y))) {
3     stop("Both arguments must be numeric")
4   }
5
6   mean_x <- sum(vec_x)/length(vec_x)
7   mean_y <- sum(vec_y)/length(vec_y)
8
9   numerator <- sum((vec_x - mean_x) * (vec_y - mean_y))
10  denominator <- (
11    sum((vec_x - mean_x)^2)^1/2 *
12    sum((vec_y - mean_y)^2)^1/2
13  )
14
15  r <- numerator/denominator
16
17  # Make sure that floating point arithmetic does not
18  # produce absolute values larger than 1
19  r <- max(min(r, 1.0), -1.0)
20
21  return(r)
22 }

```

R debugger

- In addition to simply using `print()` function, R offers an interactive source code debugger.
- It lets you
 - Step through the function at its execution time
 - Check the internal state as well as
 - Run arbitrary code in that context
 - Set breakpoints when execution pauses for inspection



Extra: Debugging with the RStudio IDE

R Debugging Facilities

- Three functions provide the the main entries into the debugging mode:
 - `browser( )` - pauses the execution at a dedicated line in code (breakpoint)
 - `debug( )/undebug( )` - (un)sets a flag to run function in a debug mode (setting through)
 - `debugonce( )` - triggers single stepping through a function

Breakpoints

```
1 calculate_median <- function(a) {  
2   a <- sort(a)  
3   n <- length(a)  
4   m <- (n + 1) %/% 2  
5   if (n %% 2 == 1) {  
6     med <- a[m]  
7   } else {  
8     browser()  
9     med <- mean(a[m:m+1])  
10  }  
11  return(med)  
12 }
```

```
1 ## Example for running in RStudio  
2 v2 <- c(0, 1, 2, 2)  
3 calculate_median(v2)
```

Called from: calculate_median(v2)

debug: med <- mean(a[m:m + 1])

debug: return(med)

[1] 2

Debugger Commands

Command	Description
<code>n(ext)</code>	Execute next line of the current function
<code>s(tep)</code>	Execute next line, stepping inside the function (if present)
<code>c(ontinue)</code>	Continue execution, only stop when breakpoint is encountered
<code>f(inish)</code>	Finish execution of the current loop or function
<code>Q(uit)</code>	Quit from the debugger, executed program is aborted

debugonce()

- `debugonce( )` function allows to run and step through the function.
- It is equivalent to setting a breakpoint at the first line of the function and then running it.

`debugonce(<function_name>, <arg_1>, <arg_2>, ..., <arg_n>)`

```
1 ## Example for running in RStudio
2 debugonce(calculate_median)
3 calculate_median(v2)
```

debugging in: calculate_median(v2)

```
debug: {
  a <- sort(a)
  n <- length(a)
  m <- (n + 1)%/%2
  if (n%%2 == 1) {
    med <- a[m]
  }
  else {
    browser()
    med <- mean(a[m:m + 1])
  }
  return(med)
}
[1] 2
```

Exercise: Using Debugger

- Let's look again at the problematic `calculate_median` function from the lecture.
- Run R debugger and step through it.
- While inside the function print out the values of `m` and the result of summation.
- Fix the bugs.

Week 5 Exercise (unassessed)

- Create tests for the fixed version of `pearson()` function that:
 - Check that the length of the result is 1;
 - Check that the result is between -1 and 1;
 - Check that the function errors out on non-numeric inputs.