Week 5 Tutorial: Debugging and Testing in R

POP77001 Computer Programming for Social Scientists

Module website: tinyurl.com/POP77001

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: Debug function for Pearson correlation

- See the function for calculating Pearson correlation below.
- Recall that sample correlation can be calsulated 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

In [2]:

```r
pearson <- function(vec_x, vec_y) {
  if (!(is.numeric(vec_x) && is.numeric(vec_y))) {
    stop("Both arguments must be numeric")
  }

  mean_x <- sum(vec_x)/length(vec_x)
  mean_y <- sum(vec_y)/length(vec_y)

  numerator <- sum((vec_x - mean_x) * (vec_y - mean_y))
  denominator <- (
    sum((vec_x - mean_x)^2)^1/2 *
      sum((vec_y - mean_y)^2)^1/2
  )

  r <- numerator/denominator

  # Make sure that floating point arithmetic does not
  # produce absolute values larger than 1
  r <- max(min(r, 1.0), -1.0)

  return(r)
}
```

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 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

Breakpoints

In [3]:

```
calculate_median <- function(a) {
  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)
}
```

Breakpoints

In [3]:
```
calculate_median <- function(a) {
  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)
}
```

In [4]:
```
## Example for running in RStudio
v2 <- c(0, 1, 2, 2)
calculate_median(v2)
```

```
Called from: calculate_median(v2)
debug at <text>#9: med <- mean(a[m:m + 1])
debug at <text>#11: return(med)

[1] 2
```

Common debugger commands

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

Debug a function

- `debugonce()` function allows to run and step through the function

```
debugonce(<function_name>, <*args>, <**kwargs>)
```

In [5]:

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

```
debugging in: calculate_median(v2)
debug at <text>#1: {
    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)
}
debug at <text>#2: a <- sort(a)
debug at <text>#3: n <- length(a)
debug at <text>#4: m <- (n + 1)%/%2
debug at <text>#5: if (n%%2 == 1) {
    med <- a[m]
} else {
    browser()
    med <- mean(a[m:m + 1])
}
debug at <text>#8: browser()
debug at <text>#9: med <- mean(a[m:m + 1])
```

```
debug at <text>#11: return(med)
exiting from: calculate_median(v2)

[1] 2
```

Exercise: Use built-in debugger to fix a function

- 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 `pearson()` and `calculate_median()` functions that
    - Test whether the sign of a calculated pearson correlation is correct
    - Test whether median calculated on an array with even number of elements has an absolute difference of no more than 0.0001 from the correct answer