Introduction to Debugging

Kirsten

12 sep 2018

This document contains a short introduction to basic debugging methods in Rstudio. The describtion is based on [HW] (http://adv-r.had.co.nz/Exceptions-Debugging.html#debugging-techniques (d. 4 sep 2018)). In the following,

- (T:) implies that you should try somthing out, and a result wil not always be explained.
- (NOTE:) will be used for things you should pay attention to or be aware about, and
- (AI:) will be used for additional information not discussed further in this document.

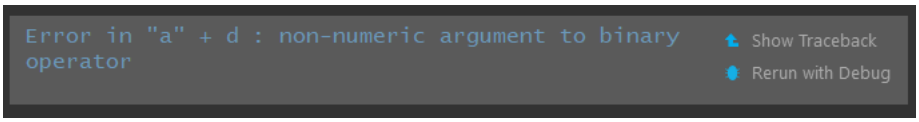
Error

We start by introducing code that creates an error. You should be able to locate the error quite easily.

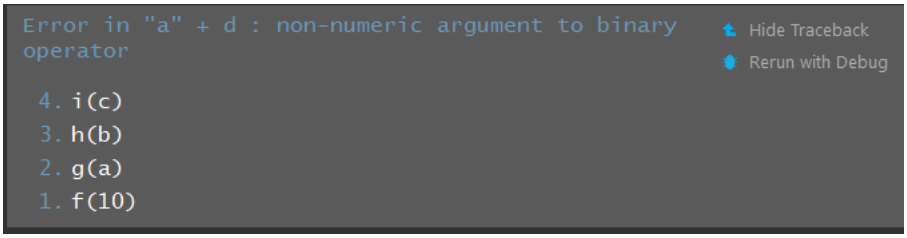
```
f <- function(a) g(a)
g <- function(b) h(b)
h <- function(c) i(c)
i <- function(d) "a" + d
f(10)
```

```
## Error in "a" + d: non-numeric argument to binary operator
```

You should see the following in your console



Choose `Show Traceback` in console (if not visable go to `Debug -> On error -> Error inspect`) or write `traceback()` to see where R runs into an error. This will give you the following output

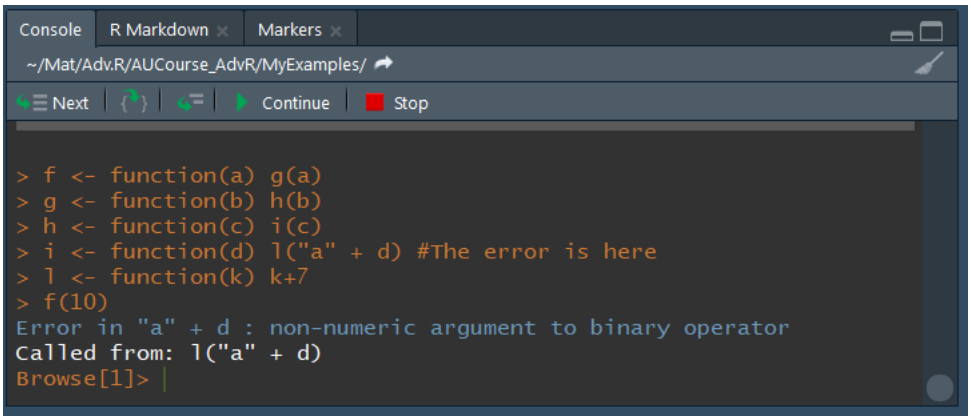


When you see this try to choos rerun with debug instead. You should get something like the following picture.

You can write the following comand in `browser()` mode

- n: execute next step
- s: step into function (if next step is a function)
- f: finish to execute current loop or function
- c: continue leave debugging and continue executing the function
- Q: quit and terminate the function and return to global workspace

For more options see [HW]



If you want to print a variabel that has the same name as a command use `print(variable)`. F.eks note the difference between writing `print(c)` and `c` in debugging mode.

Enter a function at a pre-specified line

Write the following code, and adde the `browser()` comand.

```
fac <- function(x){
  y <- 1
  browser()
  for (n in 1:x){
    y <- y*n
  }
  return(y)
}
fac(4)
```

- T1: Now try to go through the function using different commands for browser. Pay attention to variable names.
- T2: Try to create a function containing a function to see the difference between n and s.
- T3: A simpler situation using Rstudio, to try this out open the file 'Debugging R studio'

Add browser using functions

Run the following code

```
fac1 <- function(x){
  y <- 1
  for (n in 1:x){
    y <- y*n
  }
  return(y)
}
```

and try out the function `debugonce()` and `debug()`.

```
debugonce(fac1)
fac(5)

debug(fac1) #Insert browser()
```

```r
fac1(4)
fac1()
undebug(fac1) #Remove browser again
```

We can do the same with functions we cannot source

```r
debugonce(sample)
sample(4)

debug(mean)
mean(1:7) #see how the mean works
mean()
undebug(mean)
```

- T4: Try this out with one of your own functions or another existing R-function.
- NOTE: when using `debug()` remember to `undebug()` again.
- AI: If you have a specific file and line number in mind you can use: `utils::setBreakpoint()`. Check out the function yourself. The `::` and `:::` functions is used to acces exported and internal variables (See help page).

Setting error options

`option()` is a function that take various arguments, as the help page illustrate. Here we shall only cover some options related to debugging and errors.

Write `options(error=browser)`and try write `fac1(2)` notice, that nothing happens. If we make an error by writing

```r
fac2 <- function(x){
  y <- 1
  for (i in 1:x){
    y <- y*n
  }
  return(y)
}
fac2(2)
```

we enter into debugging mode. To reset error options to default value `NULL`, write `options(error = NULL)`.

- NOTE: before setting the options, note what the default settings is, to be able to go back to default settings.

Now try to write `options(error = recover)`and then run `fac2(2)`. In the console you are asked to `enter a frame number, or 0 to excit`. Write 1 when R writes selection: in the console to enter the function. Since the function contains only one function you cannot write other numbers at selection. When you get a browse response enter `objects()` to see the object in the function. You can see the value of the objects as before.

- T: try this out with the function f used earlier, can you select other items?

Again you can reset to default by writing 'options(error = NULL)

- AI: In the [HW] you find a function that reset `error=NULL` after one debugging once. Sometimes your program returns a warning and not an error. This might og might not effect your specific goal, but it means that R tryed to corrected a mistake. Since errors are esier to find using debugging, you can turn warnings into errors by writing `options(warn = 2)`.

An example of Error handling

Run the following code. As expected we see an error message

```
f1 <- function(x) {
  log(x)
  10
}
f1("x")
```

```
## Error in log(x): non-numeric argument to mathematical function
```

Now try to run the following

```
f1 <- function(x) {
  try(log(x))
  10
}
f1("x")
```

```
## [1] 10
```

Do you notice a difference? You should see that the second version continue to run the function even though we see an error.

- T: try to replace `try(log(x))` in the code above with `try(log(x), silent=TRUE)`. What do you expect to happen? What happens?

The `try()` function can be used on multiple lines, fill out the following code with somthing yourself

```
try({
  ## T: write some code with an error here
})
```

The `class()` function can tell you if your trial succeded or failed.

```
success <- try(1 + 2)
failure <- try("a" + "b")
class(success)
```

```
## [1] "numeric"
```

- AI: accord to [HW] there is no build in function to handle the class of errors. Here is a function that does just that (For more details see [HW]). You should try to run the code and identify the difference

```
is.error <- function(x) inherits(x, "try-error")
elements <- list(1:10, c(-1, 10), c(TRUE, FALSE), letters)

results <- lapply(elements, log)
succeeded <- !vapply(results, is.error, logical(1))
succeeded

results <- lapply(elements, function(x) try(log(x)))
succeeded <- !vapply(results, is.error, logical(1))
succeeded

str(results[succeeded])
str(elements[!succeeded])
```