Debugging in RStudio*

Jackson Hoffart

* Adapted from Jennifer Bryan's "What they Forgot to Teach You About R"

Finding your bug is a process of confirming the many things that you believe are true — until you find one which is not true.

—Norm Matloff

debugging

- what do we do when code doesn't work as intended?
- some bugs can be easy to see/ fix
- some bugs appear only after many layers of calls
  - hard to find
  - hard to fix
- debugging is a **VAST** topic

debugging strategies in RStudio

- traceback() – to determine where a given error is occurring
- print(), str(), cat() or message() statement – output diagnostic information in code
- browser() – open an interactive debugger before an error
- debug() – automatically open a debugger at the start of a function call
- trace() – start a debugger at a location inside the function
  - (666th iteration of a loop for example)

debugging strategies in RStudio

- traceback() – to determine where a given error is occurring
- print(), str(), cat() or message() statement – output diagnostic information in code
- browser() – open an interactive debugger before an error
- debug() – automatically open a debugger at the start of a function call
- trace() – start a debugger at a location inside the function

traceback()

- prints the call-stack of the last uncaught error
- useful when an error message isn’t helpful

traceback()

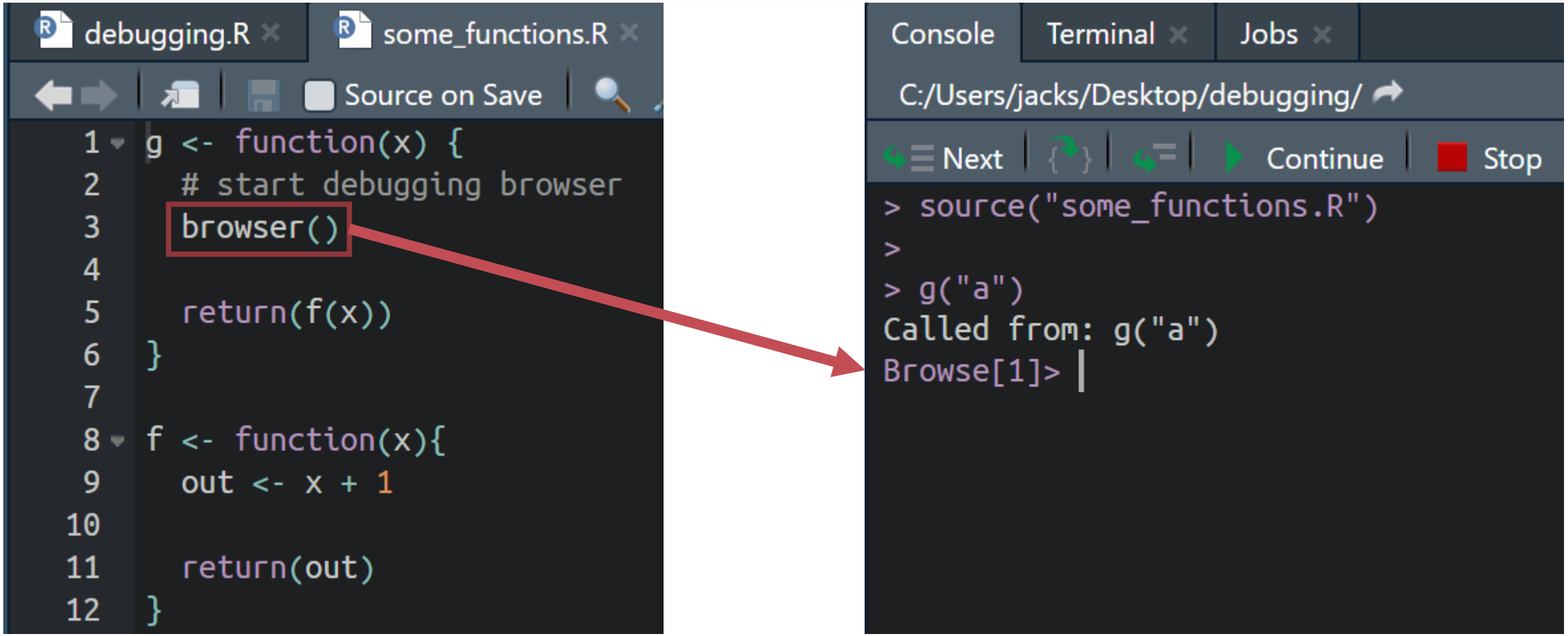
```
> # a simple example of a multiple level function-call
> f <- function(x) x + 1
> g <- function(x) f(x)
>
> # bugger
> g("a")
Error in x + 1 : non-numeric argument to binary operator
>
> traceback()
2: f(x) at #1
1: g("a")
>
```

browser()

- added directly to the source code*
- stops execution from wherever it is called
- open's R's interactive (and powerful) debugger:
  - run any R command from "within the function"
  - access objects available in the function's **current** environment

*need to have access to the source code!

browser()


debugging.R
some_functions.R
Source on Save
1 g <- function(x) {
2 # start debugging browser
3 browser()
4
5 return(f(x))
6 }
7
8 f <- function(x){
9 out <- x + 1
10
11 return(out)
12 }
Console
Terminal
Jobs
C:/Users/jacks/Desktop/debugging/
Next
Continue
Stop
> source("some_functions.R")
>
> g("a")
Called from: g("a")
Browse[1]>

browser() syntax

- ls() - to determine objects in **current** environment
- str(), print(), etc. – inspect objects in the **current** environment
- n - to evaluate the next statement
- s - to evaluate the next statement, but step into the function calls
- where - to print a stack trace
- c - to leave the debugger and continue execution
- Q - to exit the debugger and return to the R prompt

debug() & debugonce()

- similar to browser()
- debug() et al. open a debugger instance
- instead of embedding in function, you set a “debug flag”
- useful if you don’t have the source code
- debug() sets this flag permanently (call undebug() to turn off)
- debugonce() sets flag only for the next time it is called

debug() & debugonce()

- similar to browser()
- debug() et al. open a debugger instance
- instead of embedding in function, you set a “debug flag”
- useful if you don’t have the source code
- debug() sets this flag permanently (call undebug() to turn off)
- debugonce() sets flag only for the next time it is called

resources

- https://rstats.wtf/debugging-r-code.html
- https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio