

Introduction to Programming with R

Dries Debeer & Benjamin Becker

31. September and 01. October 2021

Zurich University

Agenda

Day 1

- Basic Elements & Data Types
- Flow & Conditional Programming
- Loops & Iteration
- Functions (Introduction)

Day 2

- Functions (Advanced)
- Debugging
- Efficient Programming

Open questions from day 1?

Functions II

Default arguments

What happens if the user omits an argument?

```
add_things_def <- function(x) {  
  x + 10  
}  
add_things_def()
```

```
# Error in add_things_def(): argument "x" is missing,  
with no default
```

Default arguments

Default arguments are made for such instances!

```
add_things_def <- function(x = 1) {  
  x + 10  
}  
add_things_def()
```

```
[1] 11
```

Lazy Evaluation

Sometimes missing arguments are irrelevant!

```
add_things3 <- function(x, y) {  
  x + 10  
}  
add_things3(2)
```

```
[1] 12
```

Functions usually return a single object.

- (Standard) The last evaluated object
- Object defined by `return()`
- An error via `stop()`
- Additional: Warnings + Messages

Output

```
return()
```

```
add_things_return <- function(x = 1) {  
  x2 <- x*2  
  return(x)  
  out <- x + x2  
  out  
}  
add_things_return(2)
```

```
[1] 2
```

Output

Error: stop()

```
add_things_stop <- function(x = 1) {  
  x2 <- x*2  
  stop("My own error message")  
  out <- x + x2  
  out  
}  
add_things_stop(2)
```

```
# Error in add_things_stop(2): My own error message
```

Output

Warnings: warning()

```
add_things_warning <- function(x = 1) {  
  x2 <- x*2  
  warning("My own warning message")  
  out <- x + x2  
  out  
}  
add_things_warning(2)
```

```
# Warning in add_things_warning(2):  My own warning  
message
```

```
[1] 6
```

Output

Messages: message()

```
add_things_message <- function(x = 1) {  
  x2 <- x*2  
  message("My own message")  
  out <- x + x2  
  out  
}  
add_things_message(2)
```

My own message

[1] 6

Writing Functions

Before creating the function

- What should my function do?
- Input (Arguments)
- Output

After creating the function

- Test it
- Add input validation
- Document it

Functions III

Functions III

- Good functions?
- dot dot dot
- `on.exit()`
- Accessing the call

What makes a good function?

Pure functions!

- no side effects
- the only output is returned
- no dependency on global environment
- only input via arguments

Results in easier understanding and higher portability.

How can functions receive flexible numbers of inputs?

Examples:

- `sum()`
- `save()`
- ...

via dot dot dot (...)

```
add_all_things2 <- function(...) {  
  l <- list(...)  
  do.call(sum, l)  
}  
add_all_things2(2, 3, 5, 10)
```

[1] 20

on.exit()

Performing an action when the function terminates

```
add_things <- function(x, y) {  
  on.exit(cat("Sum of", x, "and", y))  
  x <- x + 20  
  x + y  
}  
out <- add_things(1, 2)
```

Sum of 21 and 2

```
out
```

```
[1] 23
```

Accessing the function call

Accessing the function call

```
showArgs <- function(x, y) {  
  match.call()  
}  
showArgs(1, 2)
```

```
showArgs(x = 1, y = 2)
```

Debugging

Debugging

- `browser()`
- `traceback()`
- `options(error = recover)`
- `options(warn = 2)`

Also:

- `trace()` & `untrace()`
- `debug()` & `undebug()`, `debugonce()`

browser()

Inspecting a function interactively

```
some_function <- function(x, y) {  
  z <- x + y  
  browser()  
  z  
}  
some_function(x = 1, y = 5)
```

browser()

```
> some_function <- function(x, y) {  
+   z <- x + y  
+   browser()  
+   z  
+ }  
> some_function(x = 1, y = 5)  
Called from: some_function(x = 1, y = 5)  
Browse[1]> |
```


browser()

Navigating within a browser:

ls() Show existing objects in the current environment

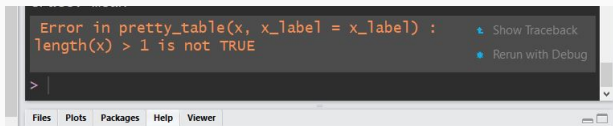
c Exit the browser and continue execution

Q Exit the browser, return to top level

where Show call stack

traceback()

Understanding the call stack:



The screenshot shows an R console window with a dark background. The error message displayed is: `Error in pretty_table(x, x_label = x_label) :
length(x) > 1 is not TRUE`. To the right of the error message are two interactive options: `Show Traceback` and `Rerun with Debug`, each preceded by a blue lightning bolt icon. Below the error message is a prompt `> |`. At the bottom of the window is a light gray toolbar with buttons for `Files`, `Plots`, `Packages`, `Help`, and `Viewer`.

traceback()

Understanding the call stack:

```
Error in pretty_table(x, x_label = x_label) :  
length(x) > 1 is not TRUE  
13. stop(simpleError(msg, call = if (p <- sys.parent(1L)) sys.c  
    all(p)))  
12. stopifnot(length(x) > 1)  
11. pretty_table(x, x_label = x_label)  
10. pretty_statistics(sub_dat$cy1, x_label = "cy1")  
9. FUN(data[x, , drop = FALSE], ...)  
8. FUN(x[[i]], ...)  
7. lapply(X = ans[index], FUN = FUN, ...)  
6. tapply(seq_len(32L), list('mtcars$carb' = c(4, 4, 1, 1, 2,  
    1,  
    4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2, 2, 4, 2, 1,  
    2,  
    2, 4, 6, 8, 2)), function (x)  
    FUN(data[x, , drop = FALSE], ...), simplify = TRUE)  
5. eval(substitute(tapply(seq_len(nd), IND, FUNx, simplify = s  
    implify)),  
    data)  
4. eval(substitute(tapply(seq_len(nd), IND, FUNx, simplify = s  
    implify)),  
    data)  
3. structure(eval(substitute(tapply(seq_len(nd), IND, FUNx, si  
    mply = simplify)),  
    data), call = match.call(), class = "by")  
2. by.data.frame(mtcars, mtcars$carb, function(sub_dat) {  
    pretty_statistics(sub_dat$cy1, x_label = "cy1")  
  })  
1. by(mtcars, mtcars$carb, function(sub_dat) {  
    pretty_statistics(sub_dat$cy1, x_label = "cy1")  
  })  
> |
```

Being able to chose an environment from the call stack:

```
# on
options(error = recover)

# off
options(error = NULL)
```

Being able to choose an environment from a call stack:

```
Error in pretty_table(x, x_label = x_label) : length(x) > 1 is not TRUE
Enter a frame number, or 0 to exit

1: by(mtcars, mtcars$carb, function(sub_dat) {
  pretty_statistics(sub_dat$cy1, x_label = "Cy1")
2: by.data.frame(mtcars, mtcars$carb, function(sub_dat) {
  pretty_statistics(sub_dat$cy1, x_label = "Cy1")
3: structure(eval(substitute(tapply(seq_len(nd), IND, FUNx, simplify = simplify))), data), call =
4: eval(substitute(tapply(seq_len(nd), IND, FUNx, simplify = simplify))), data)
5: eval(substitute(tapply(seq_len(nd), IND, FUNx, simplify = simplify))), data)
6: tapply(seq_len(32), list(mtcars$carb), c(4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4,
7: tapply(X = ans[index], FUN = FUN, ...)
8: FUN(X[[i]], ...)
9: FUN(data[x, , drop = FALSE], ...)
10: #2: pretty_statistics(sub_dat$cy1, x_label = "Cy1")
11: #3: pretty_table(x, x_label = x_label)
12: #2: stopifnot(length(x) > 1)

Selection: |
```

Warnings

Turning warnings into errors

```
# on
options(warn = 2)

# off
options(warn = 1)
```

Exercises



Efficient Programming

Wrap Up

General Advice

- Investing time in learning R pays off
- It's a steady learning curve
- Learn from masters
- Rewrite important code - the first attempt is usually not the best approach

General R Advice

- Document well
- Use a consistent style
- Write functions
- Split long functions in smaller ones
- Write wrappers
- Use Iteration (don't copy paste)
- Use matrix operations and vectorized functions instead of loops
- Use git

R Resources

- Advanced R Ed. 1 (<http://adv-r.had.co.nz/>)
- Advanced R Ed. 2 (<https://adv-r.hadley.nz/>)
- R Inferno (https://www.burns-stat.com/pages/Tutor/R_inferno.pdf)
- R Packages (<https://r-pkgs.org/>)
- Clean Code (<https://enos.itcollege.ee/~jpoial/oop/naited/Clean%20Code.pdf>)

Thank you for your attention!

Thank you for your attention!

Questions? Remarks?