

Statistics 360: Advanced R for Data Science

Lecture 1

Brad McNeney

Course Objectives

R objects: names and values

Course Objectives

Course objectives

- ▶ Work through the book Advanced R by Hadley Wickham:
<https://adv-r.hadley.nz/index.html>
- ▶ R objects: names and values
- ▶ Basic data structures and programming.
 - ▶ vectors, subsetting, control flow, functions, environments
 - ▶ No tidyverse this time
- ▶ Object-oriented programming in R
- ▶ Code performance: debugging, profiling, memory, calling Python, C or ++ from R
- ▶ Parallelizing R code (if time permits)

Getting started with R, RStudio and git

- ▶ Follow the “getting started” instructions on the class canvas page to get set up with R, RStudio and git.
 - ▶ R and RStudio will be familiar, but you may not have used git before, so leave some time for that.
- ▶ Please try to get R and RStudio installed and create an RStudio project linked to the class GitHub repository as soon as possible.
- ▶ Those still having trouble after the weekend should ask our TA, Pulindu, for help during the first lab session on Jan. 21.

Reading

- ▶ Welcome, Preface and Chapter 1 of the text.

R objects: names and values

R objects

- ▶ In R, data structures and functions are all referred to as “objects”.
- ▶ Objects are created with the assignment operator `<-`; e.g.,
`x <- c(1,2,3)`.
 - ▶ The objects a user creates from the R console are contained in the user’s workspace, called the global environment.
 - ▶ Use `ls()` to see a list of all objects in the workspace.
 - ▶ Use `rm(x)` to remove object `x` from the workspace.

Names and values

- ▶ Reference: text, chapter 2
- ▶ The description on the previous slide of `x <- c(1,2,3)` is an over-simplification.
- ▶ It is more accurate to say we've done two things: (i) created an object in computer memory that contains 1, 2, 3 and (ii) "bound" that object to the "name" `x`.

```
# install.packages("lobstr")  
library(lobstr)  
x <- c(1,2,3)  
ls()
```

```
## [1] "x"
```

```
obj_addr(x) # changes every time this code chunk is run
```

```
## [1] "0x55eafe2a19f8"
```

Syntactic vs non-syntactic

- ▶ Valid, or “syntactic” names in R can consist of letters, digits, . and _ but should start with a letter.
- ▶ Names that start with . are hidden from directory listing with `ls()`.
- ▶ Names that start with _ or a digit are non-syntactic and will cause an error.
- ▶ If you need to create or access a non-syntactic name, use backward single-quotes (“backticks”).

```
x <- 1  
.x <- 1  
`_x` <- 1  
ls()
```

```
## [1] "_x" "x"
```

Modifying, copying, binding

- Modifying a variable causes a copy to be made.

```
x <- c(1,2,3); y <- x  
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafe9ab988" "0x55eafe9ab988"
```

```
x[[2]] <- 10 # Note: x[2] <- 10 has the same effect  
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafd9712d8" "0x55eafe9ab988"
```

```
x
```

```
## [1] 1 10 3
```

```
y
```

```
## [1] 1 2 3
```

Tracing copying

- ▶ The `tracemem()` function marks an object so that a message is printed whenever a copy is made.

```
x <- c(1,2,3)
tracemem(x)
```

```
## [1] "<0x55eafe8da2b8>"
```

```
x[[2]] <- 10
```

```
## tracemem[0x55eafe8da2b8 -> 0x55eafe28bd18]: eval eval withVis
```

```
x <- 5 # removes the trace on the object
x[[1]] <- 1
```

More on tracemem()

- ▶ As the output of `tracemem()` suggests, the trace is on the object, not the name:

```
x <- c(1,2,3)
tracemem(x)
```

```
## [1] "<0x55eafe9aab78>"
```

```
y <- x
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafe9aab78" "0x55eafe9aab78"
```

```
y[[2]] <- 10
```

```
## tracemem[0x55eafe9aab78 -> 0x55eafe92c488]: eval eval withVis
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafe9aab78" "0x55eafe92c488"
```

Function calls

- R has a reputation for passing copies to functions, but in fact the copy-on-modify applies to functions too:

```
f <- function(arg) { return(arg) }  
x <- c(1,2,3)  
y <- f(x) # no copy made  
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafe28ba48" "0x55eafe28ba48"
```

```
f <- function(arg) { arg <- 2*arg; return(arg) }  
y <- f(x) # copy made  
c(obj_addr(x),obj_addr(y))
```

```
## [1] "0x55eafe28ba48" "0x55eafde4f8b8"
```

Lists

- List elements point to objects too:

```
l1 <- list(1, 2, 3)
c(obj_addr(l1), obj_addr(l1[[1]]), obj_addr(l1[[2]]), obj_addr(l1[[3]]))

## [1] "0x55eafe9abc58" "0x55eafe143c48" "0x55eafe143c10" "0x55eafe143b
# Note: ref() will print the above, formatted, but ...
```

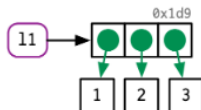


Figure 1: list

- (Note: `ref()` will print a nicely formatted version of the above, but it doesn't work with in my slides.)

Copy-on-modify in lists

- ▶ As you would expect, we only copy the list **elements** that are modified, rather than the entire list.

- ▶ `tracemem()` flags **any** change to the list

```
l1 <- list(c(1,2), c(3,4), c(5,6,7))
c(obj_addr(l1),obj_addr(l1[[1]]),obj_addr(l1[[2]]),obj_addr(l1[[3]]))

## [1] "0x55eafeba6e98" "0x55eafbccb168" "0x55eafbccb1e8" "0x55eafeba6e98"
tracemem(l1)

## [1] "<0x55eafeba6e98>"

l1[[1]] <- 55

## tracemem[0x55eafeba6e98 -> 0x55eafe94e0e8]: eval eval withVisible with
c(obj_addr(l1),obj_addr(l1[[1]]),obj_addr(l1[[2]]),obj_addr(l1[[3]]))

## [1] "0x55eafe94e0e8" "0x55eafd55fa88" "0x55eafbccb1e8" "0x55eafeba6e98"
```


Copies of lists are “shallow”

```
l2 <- l1  
l2[[3]] <- 111
```

```
## tracemem[0x55eafe94e0e8 -> 0x55eafd6f04d8]: eval eval withVisible wi  
c(obj_addr(l1),obj_addr(l1[[1]]),obj_addr(l1[[2]]),obj_addr(l1[[3]]))
```

```
## [1] "0x55eafe94e0e8" "0x55eafd55fa88" "0x55eafbccb1e8" "0x55eafeba6e  
c(obj_addr(l2),obj_addr(l2[[1]]),obj_addr(l2[[2]]),obj_addr(l2[[3]]))
```

```
## [1] "0x55eafd6f04d8" "0x55eafd55fa88" "0x55eafbccb1e8" "0x55eafed526
```

Data frames are lists ...

```
dd <- data.frame(x=1:3,y=4:6)
c(obj_addr(dd[[1]]),obj_addr(dd[[2]]))
```

```
## [1] "0x55eafbf2ec28" "0x55eafbf2ed08"
```

```
dd[,2] <- 7:9
c(obj_addr(dd[[1]]),obj_addr(dd[[2]]))
```

```
## [1] "0x55eafbf2ec28" "0x55eafc63fa90"
```

```
dd[1,] <- c(11,22)
c(obj_addr(dd[[1]]),obj_addr(dd[[2]]))
```

```
## [1] "0x55eafe8da308" "0x55eafe8da2b8"
```

```
dd[1,2] <- 111
c(obj_addr(dd[[1]]),obj_addr(dd[[2]]))
```

```
## [1] "0x55eafe8da308" "0x55eafe28b778"
```

Beware of data frame overhead

- ▶ Data frames are convenient, but the convenience comes at a cost.

- ▶ Can illustrate by tracing copying when we modify columns.

```
dd <- data.frame(x=rnorm(100)) # try yourself with rnorm(1e7)
tracemem(dd)
```

```
## [1] "<0x55eafed840a8>"
```

```
dmed <- lapply(dd,median)
```

```
## tracemem[0x55eafed840a8 -> 0x55eafd55ddd8]: as.list.data.frame as.li
```

```
dd[[1]] <- dd[[1]] - dmed[[1]] # same as dd[,1] - dmed[[1]]
```

```
## tracemem[0x55eafed840a8 -> 0x55eafebb1a00]: eval eval withVisible wi
```

```
## tracemem[0x55eafebb1a00 -> 0x55eafebb0650]: [[<-].data.frame [[<- eva
```

- Fewer copies if we do the same with a list.

```
l1 <- list(x=rnorm(100))  
tracemem(l1)
```

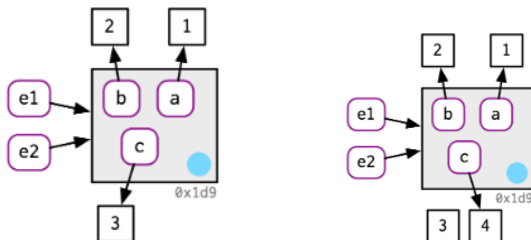
```
## [1] "<0x55eafdddfa28>"
```

```
lmed <- lapply(l1,median)  
l1[[1]] <- l1[[1]] - dmed[[1]]
```

```
## tracemem[0x55eafdddfa28 -> 0x55eafe8dea48]: eval eval withVisible wi
```

Modify-in-place

- The text claims two exceptions to the copy-on-modify, but in my experiments environments are the only one:



```
e1 <- rlang::env(a = 1, b = 2, c = 3)
e2 <- e1
e1$c <- 4
e2$c
```

```
## [1] 4
```