STAT 33B Workbook 9

Yuanrui Zhu (3034615728)

Oct 29, 2020

This workbook is due **Oct 29, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

Environments

Watch the "Environments" lecture video.

No exercises for this section.

Variable Lookup, Part 2

Watch the "Variable Lookup, Part 2" lecture video.

No exercises for this section.

The Search Path

Watch the "The Search Path" lecture video.

Exercise 1

Create a function called `locate` that finds and returns the first environment (in a chain of environments) that contains a given variable name. Your function should have a parameter `name` for the quoted variable name and a parameter `env` for the initial environment to search.

If the variable is not present in any of the environments in the chain, your function should return the empty environment.

Hint 1: Use `exists` to check whether the variable exists in `env` (and not its ancestors). If it does, return `env`. If it does not, set `env` to be its own parent and repeat this process.

Hint 2: You can use `identical` to check if two environments are equal.

YOUR ANSWER GOES HERE:

```r
locate = function(name, env) {
   if (exists(name, env, inherits = FALSE)) return (env)
   if (identical(env, emptyenv())) return (emptyenv())
   locate(name, parent.env(env))
}
```

Exercise 2

This code produces an environment `e` with several ancestors:

```r
e = new.env()
e$c = 42
e = new.env(parent = e)
e$a = "33a"
e$b = "33b"
e$c = "33ab"
e = new.env(parent = e)
e$x = 8
#e = parent.env(e)
#exists("c", e, inherits = FALSE)
identical(parent.env(parent.env(parent.env(e))), globalenv())
```

```
## [1] TRUE
```

Test your `locate` function on `e` by:

1. Locating `"c"`. Your result `result` should have `result$c` equal to `"33ab"`.
2. Locating `"zzz"`. Your result `result` should be the empty environment.
3. Locating `"e"`. Your result `result` should be the global environment.
4. Locating `"show"`. Which built-in package provides this function?

YOUR ANSWER GOES HERE:

Part 1

```r
result = locate("c", e)
result$c
```

```
## [1] "33ab"
```

Part 2

```r
result2 = locate("zzz", e)
identical(result2, emptyenv())
```

```
## [1] TRUE
```

Part 3

```r
result3 = locate("e", e)
identical(result3, globalenv())
```

```
## [1] TRUE
```

Part 4

```r
locate("show", e)
```

```
## <environment: package:methods>
## attr(,"name")
## [1] "package:methods"
## attr(,"path")
## [1] "/Library/Frameworks/R.framework/Versions/4.0/Resources/library/methods"
```

```r
# package "methods" provides this function
```

The Colon Operators

Watch the "The Colon Operators" lecture video.

No exercises for this section.

Closures

Watch the "Closures" lecture video.

Exercise 3

Recall the `find_fib2` function (from week 8) for computing Fibonacci numbers:

```
find_fib2 = function(n, fib = c(1, 1)) {
  len = length(fib)
  if (n <= len)
    return (fib[n])

  fib = c(fib, fib[len - 1] + fib[len])
  Recall(n, fib)
}

find_fib2(40)
```

```
## [1] 102334155
```

The key to computing Fibonacci numbers efficiently is to keep a record of the numbers that have already been computed. The `find_fib2` function does this by passing the record of computed numbers on through the `fib` parameter.

An alternative to passing the record through a parameter is to store the record in the function's enclosing environment. Write a function `find_fib3` that does this.

Your function should still have a parameter `n` and return the `n`-th Fibonacci number. Your function should **NOT** have a parameter `fib`.

Test your function by computing `fib(40)`. If your function is working correctly, it should be able to compute this number in less than 5 seconds, and the number should be `102334155`.

Hint 1: Create a factory function `make_find_fib3` to provide the enclosing environment. The factory function should not have any parameters, and should return your `find_fib3` function.

Note: Using the enclosing environment to store values that have been computed is called "memoization". Memoization is a useful strategy for improving efficiency in many programming problems.

YOUR ANSWER GOES HERE:

```
make_find_fib3 = function(){
 fib = c(1, 1)
 find_fib3 = function(n){
   len = length(fib)
   if (n <= len) return (fib[n])
   fib <<- c(fib, fib[len] + fib[len-1])
   Recall(n)
 }
}

fib3 = make_find_fib3()
#fib3(40)
```

Exercise 4

1. Enclosing environments persist between calls. Explain how you think this will affect the speed and memory usage of `find_fib3` compared to `find_fib2`. What are the advantages and disadvantages of the two different functions?

Hint: How long does it take to compute `find_fib3(40)` the first time? The second time? The third time?

2. Use the microbenchmark package to benchmark `find_fib2` and `find_fib3` for `n` equal to 30 and `n` equal to 40. Which function is faster?

YOUR ANSWER GOES HERE:

Part 1

```r
library(microbenchmark)
microbenchmark(fib3(40))
```

```
## Unit: nanoseconds
##      expr min  lq    mean median   uq    max neval
##  fib3(40) 911 932 3918.78   1097 1451 252607   100
```

```r
microbenchmark(fib3(40))
```

```
## Unit: nanoseconds
##      expr min     lq    mean median   uq   max neval
##  fib3(40) 982 1000.5 1436.89   1018 1230 29445   100
```

```r
microbenchmark(fib3(40))
```

```
## Unit: nanoseconds
##      expr min    lq    mean median  uq   max neval
##  fib3(40) 908 925.5 1203.57    933 972 15410   100
```

```r
# find_fib3() makes the function costing more time when reproducing,
# but it is much quicker for the first time called,
# it also cost more memory usage; find_fib2 vice versa.
```

Part 2

```r
microbenchmark(find_fib2(30))
```

```
## Unit: microseconds
##           expr    min     lq     mean  median     uq    max neval
##  find_fib2(30) 39.636 40.499 41.80768 40.9635 42.288 64.124   100
```

```r
microbenchmark(fib3(30))
```

```
## Unit: nanoseconds
##      expr min  lq    mean median   uq   max neval
##  fib3(30) 915 928 1326.93    948 1093 26291   100
```

```r
microbenchmark(find_fib2(40))
```

```
## Unit: microseconds
##           expr    min     lq     mean median      uq    max neval
##  find_fib2(40) 56.927 57.929 60.22634 58.931 61.1655 86.988   100
```

```r
microbenchmark(fib3(40))
```

```
## Unit: nanoseconds
##      expr min  lq    mean median   uq   max neval
##  fib3(40) 920 932 1561.29  944.5 1327 37260   100
```

```r
# obviously, fib3 is much faster than fib2
```