STAT 33B Homework 5

Ming Fong (3035619833)

Nov 5, 2020

This homework is due **Nov 5, 2020** by 11:59pm PT.

Homeworks are graded for correctness.

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.

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.

Exercise 1

A *stack* is a data structure that stores objects in last-in first-out order.

You can visualize a stack as a stack of cafeteria trays. When you add a tray to the stack, it goes on top and *pushes* down the other trays. When you remove a tray from the stack, it comes off the top, and the other trays *pop* up.

Unlike a list, you generally cannot access objects in a stack in arbitrary order. Instead, you can only pop off the objects in reverse of the order they were pushed onto the stack.

Write a function `make_stack` that returns a list with 2 elements:

1. An element `push` which is a closure that pushes an object onto the stack.
2. An element `pop` which is a closure that pops an object off the stack.

Make your `pop` function print an informative error message if the stack is empty.

Test your functions to make sure they work correctly. For example, this code should return 3 `TRUE`s:

```
stack = make_stack()

stack$push(3)
stack$push(4)
stack$push(5)

stack$pop() == 5
stack$pop() == 4
stack$pop() == 3
```

Hint: Use a list called ***stack_data****, local to* ***make_stack****, to store the objects on the stack. R will automatically increase the length of a list if you assign to a position beyond than the current length.*

Note: You can optionally make the stack more efficient by overallocating ***stack_data*** *and using a separate variable to keep track of the position of the "top" element.*

YOUR ANSWER GOES HERE:

```
make_stack = function() {
  stack_data = list()
  push = function(n) {
    stack_data[length(stack_data) + 1] <<- n
  }
  pop = function() {
    if(length(stack_data) == 0) {
      stop("Failed to pop, stack was empty")
    }
    ret = stack_data[[length(stack_data)]]
    stack_data[[length(stack_data)]] <<- NULL
    ret
  }
  stack = list(push = push, pop = pop)

}
```

Exercise 2

This exercise is worth 30 points instead of the usual 10 points.

The purpose of this exercise is to practice debugging.

The function in the next section has many bugs. Some of these bugs cause errors which show up when trying to parse the function, while others don't show up until trying to run the function. There are also silent bugs which will not cause errors but will lead to incorrect results.

The Function

The `tip_calculator()` function, shown below, is meant to calculate the tip and grand total for a restaurant bill. There are multiple ways to calculate these, depending on the tip rate, tax rate, and whether the tip is on the subtotal before or after tax. The function's parameters are:

- `subtotal` – the bill before taxes are added
- `percent_tip` – the percentage of the total add as a tip
- `post_tax` – if `TRUE`, calculate the tip after adding tax; otherwise, calculate the tip before adding tax
- `percent_tax` – the percentage of the total to add as tax

The function is defined as:

```
tip_calculator = function(subtotal, percent_tip = 0.20, post_tax = TRUE,
  percent_tax = 0.0925)
{
  tax = subtotal**percent_tax
  pre_tip = subtotal
  if(post_tax) {
    pre_tip = subtotal + tax
  }
  tip = pre_tip*0.0925
```

```
  grand_total == subtotal + tax + tip

  out = c("tip" = tip, "total" = grand_total)

  out
}
```

Your task is to find and fix each bug in the `tip_calculator()` function.

Do the debugging in rounds, with one round for each bug. After finding and fixing a bug, put the fixed code **in a new code chunk**. Then explain the steps you took to find and fix the bug.

You must perform at least 3 rounds of debugging, but more may be necessary to fix all of the bugs. Use the `browser()` function in at least one round (comment out the call to `browser()` in your fixed code).

You have not fixed all of the bugs until all of the pre-written tests at the end of this exercise run and return `TRUE`. Note that these tests may not be exhaustive, so you may want to add more tests of your own.

Round 1

Describe the bug, the steps you took to find the bug, and the steps you took to fix the bug. Place the fixed code in the cell below.

YOUR ANSWER GOES HERE:

The line `tax = subtotal**percent_tax` raises `subtotal` to the power of `percent_tax` when it should be multiplied. I also fixed the formatting and spacing of the code.

```
# Your fixed code after round 1 goes here.
tip_calculator = function(subtotal, percent_tip = 0.20, post_tax = TRUE,
  percent_tax = 0.0925)
{
  tax = subtotal * percent_tax
  pre_tip = subtotal
  if(post_tax) {
    pre_tip = subtotal + tax
  }
  tip = pre_tip * 0.0925
  grand_total = subtotal + tax + tip

  out = c("tip" = tip, "total" = grand_total)

  out
}
```

Round 2

Describe the bug, the steps you took to find the bug, and the steps you took to fix the bug. Place the fixed code in the cell below.

YOUR ANSWER GOES HERE:

I ran the tests and found that `grand_total == subtotal + tax + tip` does not assign the variable `grand_total`, `==` should be replaced with `=`

```
# Your fixed code after round 2 goes here.
tip_calculator = function(subtotal, percent_tip = 0.20, post_tax = TRUE,
  percent_tax = 0.0925)
{
```

```
  tax = subtotal * percent_tax
  pre_tip = subtotal
  if(post_tax) {
    pre_tip = subtotal + tax
  }
  tip = pre_tip * 0.0925
  grand_total = subtotal + tax + tip

  out = c("tip" = tip, "total" = grand_total)

  out
}
```

Round 3

Describe the bug, the steps you took to find the bug, and the steps you took to fix the bug. Place the fixed code in the cell below.

YOUR ANSWER GOES HERE:

I ran the first test and it was `FALSE`. This indicated somethingwas wrong with the `percent_tip` value. I used the debugger to check the value of `tip` and found that `percent_tip` was incorrectly replaced with a hardcoded `0.0925`. The error was at

`tip = pre_tip * 0.0925`

```
# Your fixed code after round 3 goes here.
tip_calculator = function(subtotal, percent_tip = 0.20, post_tax = TRUE,
  percent_tax = 0.0925)
{
  # browser()
  tax = subtotal * percent_tax
  pre_tip = subtotal
  if(post_tax) {
    pre_tip = subtotal + tax
  }
  tip = pre_tip * percent_tip
  grand_total = subtotal + tax + tip

  out = c("tip" = tip, "total" = grand_total)

  out
}
```

Round 4

Describe the bug, the steps you took to find the bug, and the steps you took to fix the bug. Place the fixed code in the cell below.

Note: If you need more that 4 rounds of debugging, add them after this section.

YOUR ANSWER GOES HERE:

After round 3, all tests passed as `TRUE` and all bugs appear to be fixed.

```
# Your fixed code after round 4 goes here.
```

Test Cases

Below are a few calls to `tip_calculator()`. The results are assigned to variables so that they can be used in further tests below.

```
test_a = tip_calculator(100)
test_a
```

```
##    tip  total
##  21.85 131.10
```

```
test_b = tip_calculator(100, 0.15)
test_b
```

```
##      tip    total
##  16.3875 125.6375
```

```
test_c = tip_calculator(100, 0.15, FALSE)
test_c
```

```
##    tip  total
##  15.00 124.25
```

```
test_d = tip_calculator(100, 0.15, FALSE, 0.0725)
test_d
```

```
##    tip  total
##  15.00 122.25
```

Below are tests that check the correctness of the results from the previous calls.

```
# Confirm that the default tip is more than the 15% tip
test_a["tip"] > test_b["tip"]
```

```
##  tip
## TRUE
```

```
test_a["total"] > test_b["total"]
```

```
## total
##  TRUE
```

```
# Confirm that tipping post-tax is more than tipping pre-tax
test_b["tip"] > test_c["tip"]
```

```
##  tip
## TRUE
```

```
test_b["total"] > test_c["total"]
```

```
## total
##  TRUE
```

```
# Confirm that the default tax is more than the base California tax of 7.25%
test_c["total"] > test_d["total"]
```

```
## total
##  TRUE
```

```
# Confirm that the grand total minus the tip is the subtotal plus tax
(test_a["total"] - test_a["tip"]) == 100 * 1.0925
```

```
## total
```

```
##  TRUE
```

```
(test_b["total"] - test_b["tip"]) == 100 * 1.0925
```

```
## total
##  TRUE
```

```
(test_c["total"] - test_c["tip"]) == 100 * 1.0925
```

```
## total
##  TRUE
```

```
(test_d["total"] - test_d["tip"]) == 100 * 1.0725
```

```
## total
##  TRUE
```