Week-5: Code-along

Jing Ying

2023-09-10

```
knitr::opts_chunk$set(echo = TRUE)
```

II. Code to edit and execute using the Code-along.Rmd file

A. Writing a function

1. Write a function to print a "Hello" message (Slide #14)

```
# Enter code here
# name <- "Kashif"
# print(paste0("Hello ", name, "!"))

say_hello_to <- function(name) {
 print(paste0("Hello ", name, "!"))
}

say_hello_to('Kashif')
```

```
## [1] "Hello Kashif!"
```

2. Function call with different input names (Slide #15)

```
# Enter code here

say_hello_to('Kashif')
```

```
## [1] "Hello Kashif!"
```

```
say_hello_to('Zach')
```

```
## [1] "Hello Zach!"
```

```
say_hello_to('Deniz')
```

```
## [1] "Hello Deniz!"
```

3. typeof primitive functions (Slide #16)

```
# Enter code here

typeof(`+`)
```

```
## [1] "builtin"
```

```
typeof(sum)
```

```
## [1] "builtin"
```

4. typeof user-defined functions (Slide #17)

```
# Enter code here

typeof(say_hello_to)
```

```
## [1] "closure"
```

```
typeof(mean)
```

```
## [1] "closure"
```

5. Function to calculate mean of a sample (Slide #19)

```
# Enter code here

calc_sample_mean <- function(sample_size) {
 mean(rnorm(sample_size))
 }
```

6. Test your function (Slide #22)

```
# With one input

calc_sample_mean(1000)
```

```
## [1] 0.006358833
```

```
# With vector input

calc_sample_mean(c(100, 300, 3000))
```

```
## [1] -0.2396945
```

7. Customizing the function to suit input (Slide #23)

```
# Enter code here

library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
sample_tibble <- tibble(sample_sizes =
 c(100, 300, 3000))

sample_tibble %>%
group_by(sample_sizes) %>%
mutate(sample_means =
 calc_sample_mean(sample_sizes))
```

```
## # A tibble: 3 x 2
## # Groups:   sample_sizes [3]
##   sample_sizes sample_means
##          <dbl>        <dbl>
## 1          100      -0.0789
## 2          300      -0.0368
## 3         3000       0.0144
```

8. Setting defaults (Slide #25)

```
# First define the function

calc_sample_mean <- function(sample_size,
 our_mean, our_sd)
 {
 sample <- rnorm(sample_size,
 mean = our_mean,
 mean(sample),
 sd = our_sd)
 }

# Call the function

calc_sample_mean <- function(sample_size,
 our_mean=0,
```

```
 our_sd=1) {
 sample <- rnorm(sample_size,
 mean = our_mean,
 sd = our_sd)
 mean(sample)
}

calc_sample_mean(sample_size = 10)
```

```
## [1] -0.3236144
```

9. Different input combinations (Slide #26)

```
# Enter code here

calc_sample_mean(10, our_sd = 2)
```

```
## [1] 0.3165914
```

```
calc_sample_mean(10, our_mean = 6)
```

```
## [1] 6.264243
```

```
calc_sample_mean(10, 6, 2)
```

```
## [1] 4.914621
```

10. Different input combinations (Slide #27)

```
# set error=TRUE to see the error message in the output
# Enter code here

calc_sample_mean(our_mean = 5)
```

```
## Error in calc_sample_mean(our_mean = 5): argument "sample_size" is missing, with no default
```

11. Some more examples (Slide #28)

```
# Enter code here

add_two <- function(x) {
 x+2
}

add_two(4)
```

```
## [1] 6
```

```
add_two(-34)
```

```
## [1] -32
```

```
add_two(5.784)
```

```
## [1] 7.784
```

B. Scoping

12. Multiple assignment of z (Slide #36)

```
# Enter code here

z <- 1
sprintf("The value assigned to z outside the function is %d",z)
```

```
## [1] "The value assigned to z outside the function is 1"
```

```
foo <- function(z = 2) {
 # reassigning z
 z <- 3
 return(z+3)
}
foo()
```

```
## [1] 6
```

13. Multiple assignment of z (Slide #37)

```
# Enter code here

z <- 1
foo <- function(z = 2) {
z <- 3
return(z+3)}
foo(z = 4)
```

```
## [1] 6
```

```
sprintf("The final value of z after reassigning it to a different value inside the function is %d",z)
```

```
## [1] "The final value of z after reassigning it to a different value inside the function is 1"
```