Week-5: Code-along

Soh Li Ying

2023-09-09

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)

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

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

```
say_hello_to('Kashif')
```

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

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

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

3. typeof primitive functions (Slide #16)

```
typeof(`+`)
```

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

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

```
typeof(say_hello_to)
```

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

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

```
mean(rnorm(1))
```

```
## [1] -0.003211806
```

6. Test your function (Slide #22)

```
calc_sample_mean <- function(n) mean(rnorm(n))
```

```
calc_sample_mean(2000)
```

```
## [1] -0.01376283
```

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

```
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
```

```
#create a vector to test our function
sample_tibble <- tibble(sample_sizes =
                          c(100, 300, 3000))

#using group_by to group unique entries,
# allowing calc_sample_mean
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      -0.121
## 2          300       0.0966
## 3         3000      -0.0286
```

8. Setting defaults (Slide #25)

```
# First define 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)
}

# Call the function
calc_sample_mean(sample_size = 10)
```

```
## [1] -0.1647187
```

9. Different input combinations (Slide #26)

```
calc_sample_mean(10, our_sd = 2)
```

```
## [1] 0.1497353
```

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

```
## [1] 9.832145
```

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

```
## [1] 5.407362
```

10. Different input combinations (Slide #27)

```
# set error=TRUE to see the error message in the output
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)

```
# Add 2 to input x
add_two <- function(x) {
  x+2
}

add_two
```

```
## function(x) {
##   x+2
## }
```

```
add_two(-34)
```

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

B. Scoping

12. Multiple assignment of z (Slide #36)

```
# Initialize z
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"
```

```
#declare a function, passing value of 2 for z
addition_of_three <- function (z = 2) {
  #reassigning z
  z <-3
  return(z+3)
}

addition_of_three()
```

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

13. Multiple assignment of z (Slide #37)

```
# Initialize z
z <- 1

#declare a function, passing value of 2 for z
addition_of_three <- function(z = 2){
  #reassigning z
  z <- 3
  return(z+3)
}

#another reassingment of z
addition_of_three(z = 4)
```

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

```r
#Accessing z outside of the function
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"
```