Week-5: Code-along

Ho Wei Ni

2023-09-10

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("Panaree")
```

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

```
say_hello_to("Zi Qi")
```

```
## [1] "Hello Zi Qi!"
```

```
say_hello_to("Kymie")
```

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

3. typeof primitive functions (Slide #16)

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

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

```
typeof(sum)
```

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

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

```
typeof(say_hello_to)
```

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

```
typeof(mean)
```

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

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

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

6. Test your function (Slide #22)

```
calc_sample_mean(200)
```

```
## [1] -0.04704704
```

```
calc_sample_mean(c(100,234,2987))
```

```
## [1] -0.6141175
```

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
```

```r
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      0.00692
## 2          300      0.0340
## 3         3000      0.0256
```

8. Setting defaults (Slide #25)

```r
# 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] 0.3209066
```

9. Different input combinations (Slide #26)

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

```
## [1] 0.03140847
```

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

```
## [1] 10.58723
```

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

```
## [1] 5.238723
```

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 rnorm(sample_size, mean = our_mean, sd = our_sd): argument "sample_size" is missing, with n
```

11. Some more examples (Slide #28)

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

add_two(250)
```

```
## [1] 252
```

```
divide_two <- function(x) {
  x/2
}

divide_two(250)
```

```
## [1] 125
```

B. Scoping

12. Multiple assignment of z (Slide #36)

```
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)

```r
z <- 1

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

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

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

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