

Week-5: Code-along

WU YUTONG

2023-09-12

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

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))
#}

#alternatively,
calc_sample_mean <- function(sample_size) {
  random_sample <- rnorm(sample_size)
  sample_mean <- mean(random_sample)
  return(sample_mean)
}
```

6. Test your function (Slide #22)

```
# With one input
calc_sample_mean(1000)
```

```
## [1] 0.01681538
```

```
# With vector input
calc_sample_mean(c(100,300,1000))
```

```
## [1] -0.1589061
```

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

```
# Enter code here
library(tidyverse)
```

```
## — Attaching core tidyverse packages — tidyverse 2.0.0 —
## ✓ dplyr      1.1.2    ✓ readr      2.1.4
## ✓ forcats    1.0.0    ✓ stringr   1.5.0
## ✓ ggplot2     3.4.3    ✓ tibble     3.2.1
## ✓ lubridate  1.9.2    ✓ tidyr      1.3.0
## ✓ purrr       1.0.2
## — Conflicts — tidyverse_conflicts() —
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ! 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 × 2
## # Groups:   sample_sizes [3]
##   sample_sizes sample_means
##   <dbl>         <dbl>
## 1      100      -0.0613
## 2      300       0.0356
## 3     3000       0.0217
```

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] -0.2043708
```

9. Different input combinations (Slide #26)

```
# Enter code here
calc_sample_mean(10, our_sd = 2)
```

```
## [1] -0.4870942
```

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

```
## [1] 6.160105
```

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

```
## [1] 4.570476
```

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 rnorm(sample_size, mean = our_mean, sd = our_sd): argument "sample_size" is missing, with no default
```

11. Some more examples (Slide #28)

```
# Enter code here
add_two <- function(x) {
  y+2
}
add_two(4)
```

```
## Error in add_two(4): object 'y' not found
```

```
add_two <- function(x) {
  x+2
}
add_two(4)
```

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

```
add_numbers <- function(x,y) {
  x+y
}
add_numbers(45, 12)
```

```
## Error in add_numbers(45, 12): could not find function "add_numbers"
```

```
add_numbers <- function(x,y) {
  x+y
}
add_numbers(45, 12, 72)
```

```
## Error in add_numbers(45, 12, 72): unused argument (72)
```

```
add_numbers <-function(x,y) {
  z = 20
  x+y
  return(x+y, z)
}
add_numbers(1,2)
```

```
## Error in return(x + y, z): multi-argument returns are not permitted
```

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) {
  #reassigning z
  z <- 3
  return(z+3)
}
#another reassignment of z
foo(z = 4)
```

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

```
#assessing z outside 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"
```