Week-5: Code-along

Marzuki Nooranas

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)

# Enter code here  
say\_hello\_to <- function(name) {  
 print(paste0 ("Hello ", name, "!"))  
}

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

# Enter code here  
myname <- "Marzuki"  
say\_hello\_to('Kashif')

## [1] "Hello Kashif!"

say\_hello\_to('Zach')

## [1] "Hello Zach!"

say\_hello\_to('Denis')

## [1] "Hello Denis!"

say\_hello\_to(myname)

## [1] "Hello Marzuki!"

### 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] -0.07142414

# With vector input  
calc\_sample\_mean(c(100, 300, 3000))

## [1] -0.2059102

### 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()  
## 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 × 2  
## # Groups: sample\_sizes [3]  
## sample\_sizes sample\_means  
## <dbl> <dbl>  
## 1 100 -0.0146  
## 2 300 -0.0899  
## 3 3000 -0.0204

### 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] -0.1916152

### 9. Different input combinations (Slide #26)

# Enter code here   
calc\_sample\_mean(10, our\_sd = 2)

## [1] 0.4712587

calc\_sample\_mean(10, our\_mean = 6)

## [1] 5.935472

calc\_sample\_mean(10, 6, 2)

## [1] 5.862578

### 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) {   
z <- 3  
return(z+3)  
}   
foo()

## [1] 6

### 13. Multiple assignment of z (Slide #37)

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"