Week-5: Code-along

Sherica Chua

`10/09/2023

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
print("Hello")
```

```
## [1] "Hello"
```

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

```
# Enter code here
function(name)
  {print(paste0("Hello",name, "!"))}
```

```
## function(name)
##   {print(paste0("Hello",name, "!"))}
```

3. typeof primitive functions (Slide #16)

```
# Enter code here
typeof(sum)
```

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

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

```
# Enter code here
typeof(mean)
```

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

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

```
# Enter code here
function(n) mean(rnorm(n))
```

```
## function(n) mean(rnorm(n))
```

6. Test your function (Slide #22)

```
# With one input
calc_sample_mean <-
function(n) mean(rnorm(n))
calc_sample_mean(1000)
```

```
## [1] 0.03334407
```

```
# With vector input
calc_sample_mean(c(
100
,
300
,
3000
))
```

```
## [1] 0.4597503
```

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 becom
e 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      0.174
## 2          300      0.0501
## 3         3000      0.00725
```

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] 0.06057801
```

9. Different input combinations (Slide #26)

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

```
## [1] 0.2519293
```

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

```
## [1] 6.243419
```

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

```
## [1] 4.787341
```

10. Different input combinations (Slide #27)

```
## 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
```

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"
```

```
# Enter code here
foo <-function(z =2)
  {
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
```