Week-5: Code-along

Janelle Tan

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

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"
```

alt way in word via unnesting function for ease of understanding

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

```
# Enter code here
calc_sample_mean <- function(sample_size) {
 mean(rnorm(sample_size))
 }
```

1. identify the constant in the prob we're solving: instruction of the command being repeated -> mean(rnorm(sample_size))

2. identify input of argument -> sample size

3. no need return

4. put within {}

5. make sure hv function(argument)

6. assign a name

6. Test your function (Slide #22)

basically calling your function

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

```
## [1] 0.01558658
```

```
# With vector input
# read ?rnorm to understand how rnorm
# inteprets vector input.
calc_sample_mean(c(100, 300, 3000))
```

impt to know what the expected types of inputs & outputs of a functions are eg vector?

```
## [1] 0.1799929
```

If we don't want to change our function, but we want to use it to deal with vectors,

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 be
come errors
```

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

#using rowwise groups the data by row,
# allowing calc_sample_mean
sample_tibble %>%
group_by(sample_sizes) %>%
mutate(sample_means =
 calc_sample_mean(sample_sizes))
```

Create tibble w column name called sample-Size assigned to vector shown

tibble is like a list, but unlike a list, all the columns/var shld hv the same no. of entries

group_by from tidyverse package

```
## # A tibble: 3 × 2
## # Groups:   sample_sizes [3]
##   sample_sizes sample_means
##          <dbl>        <dbl>
## 1          100     -0.0479
## 2          300     -0.0286
## 3         3000     -0.00284
```

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

# uses the defaults
calc_sample_mean(sample_size = 10)
```

- adding additional arguments: note commas
- note order of arguments

- Still can change value of arguments w defaults, the defaults just means that if you don't assign any values to them, default values will show

```
## [1] -0.316406
```

9. Different input combinations (Slide #26)

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

```
## [1] 0.5323534
```

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

```
## [1] 6.381824
```

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

```
## [1] 6.976762
```

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)
# BC sample size not assigned default value
```

note

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

local variable that shadows (overrides) the global variable; default in R

12. Multiple assignment of z (Slide #36)

```
# Enter code here
# 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, notice how we pass a value of 2 for z
foo <- function(z = 2) {
 # reassigning z
 z <- 3
 return(z+3)
}
foo()
```

note: bad naming convention

value defined outside the function was overridden by the value defined inside the function (z = 2),

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

which was again overridden by the reassignment of value within the function body

13. Multiple assignment of z (Slide #37)

```
# Enter code here
# Initialize z
z <- 1
# declare a function, notice how we pass a value of 2 for z
foo <- function(z = 2) {
 # reassigning z
 z <- 3
 return(z+3)
}
# another reassignment of z
foo(z = 4)
```

2nd: newly locally defined z = 3 gvs 6 as answer -> doesnt affect global z= 1

1st: z = 4 overrides specified z = 2

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

```
# Accessing z outside the function
sprintf("The final value of z after reassigning it to a different value inside the function i
s %d", z)
```

```
## [1] "The final value of z after reassigning it to a different value inside the function is
1"
```