Week-5: Code-along

Nicole Lim

2023-09-09

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)

```r
# Enter code here
print_hello <- function () { print("Hello")}

print_hello()
```

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

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

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

print_hello_to("RStudio")
```

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

```r
print_hello_to("0_0")
```

```
## [1] "Hello 0_0!"
```

3. typeof primitive functions (Slide #16)

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

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

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

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

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

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

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

```
typeof(print_hello_to)
```

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

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

```
# Enter code here
mean_of_sample <- function(samplesize) {
  mean(rnorm(samplesize))
  }
```

6. Test your function (Slide #22)

```
# With one input
mean_of_sample(50)
```

```
## [1] 0.09027672
```

```
# With vector input
?rnorm
mean_of_sample(c(100,200,700))
```

```
## [1] 1.001445
```

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

```
# Enter code here
library(tidyverse)
sample_tibble <- tibble(samplesizes = c(100, 200, 700))
#*Tibble* is a list where all the columns have the same number of entries.
#*Group_by* groups by unique entries
#*Mutate* informs new column which has entries making use of previous function
sample_tibble %>% group_by(samplesizes) %>% mutate(samplemeans = mean_of_sample(sampl
esizes))
```

```
## # A tibble: 3 × 2
## # Groups:   samplesizes [3]
##   samplesizes samplemeans
##         <dbl>       <dbl>
## 1         100      0.0610
## 2         200      0.0203
## 3         700     -0.0119
```

8. Setting defaults (Slide #25)

```
# First define the function
calc_sample_mean <- function(samplesize,our_mean=0,our_sd=1) {

  sample <- rnorm(samplesize, mean = our_mean, sd = our_sd)

  mean(sample)

}
# Call the function
calc_sample_mean(samplesize = 127)
```

```
## [1] -0.02969824
```

9. Different input combinations (Slide #26)

```
# Enter code here
calc_sample_mean(127,115,15)
```

```
## [1] 114.214
```

```
calc_sample_mean(4,12)
```

```
## [1] 11.64991
```

```
calc_sample_mean(99, our_mean = 12, our_sd = 2)
```

```
## [1] 12.05845
```

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=143)
```

```
## Error in rnorm(samplesize, mean = our_mean, sd = our_sd): argument "samplesize" is
missing, with no default
```

11. Some more examples (Slide #28)

```
# Enter code here
plus_one <- function(x) x+1
plus_one(1)
```

```
## [1] 2
```

```
plus_one("one")
```

```
## Error in x + 1: non-numeric argument to binary operator
```

```
plus_one(FALSE)
```

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

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"
```

13. Multiple assignment of z (Slide #37)

```
# Enter code here
fn <- function( z = 2 ) {
  #Reassign z
  z <- 3
  return(z+3)
}

fn(z = 5)
```

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

```
sprintf("The final value of z after reassigning it to a different value inside of the
function is %d", z)
```

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

```
#z accessed inside the function is using the value tree
#z accessed outside the function does not regard z inside the funciton
#scope defined by location of initialisation and where it can be accessed
#global scope = defined outside functions, accessed anywhere in the program
#local scope = declared inside functions, cannot be accessed outside of it
```